FACHHOCHSCHULE
GIESSEN
FRIEDBERG

Diplomarbeit
zum Thema

Management- und Web Services- Architekturen

Konzeption und Realisierung eines Überwachungssystems für Bankperipheriegeräte

Zur Erlangung des akademischen Grades
Diplom-Informatiker (FH)

vorgelegt am
Fachbereich Mathematik, Naturwissenschaften und Informatik der Fachhochschule Gießen-Friedberg

von
Roger Zacharias
Im Februar 2002

Referent: **Prof. Dr. A. H. Kaufmann**
Koreferent: **Prof. Dr. M. Scheer**

Vorwort

Technology of a sufficiently advanced level will be indistinguishable from magic. (Arthur C. Clarke)

Die vorliegende Diplomarbeit entstand im Zeitraum 10/2001 – 02/2002 an der Fachhochschule Gießen-Friedberg in Zusammenarbeit mit der Wincor Nixdorf GmbH & Co. KG unter der Betreuung von Herrn Prof. Dr. Kaufmann und Herrn Prof. Dr. Scheer. Die Aufgabenstellung ergab sich aus einem Projekt der Wincor Nixdorf GmbH & Co. KG, in welchem eine umfangreiche Überwachungslösung im Bankenbereich entwickelt wird. Dazu sollen die zurzeit neu aufkommenden Architekturen und Paradigmen untersucht und ein möglicher Prototyp erstellt werden und bei entsprechender Eignung gegebenenfalls in diese Überwachungslösung mit einfließen.

Die besondere Herausforderung dieser Diplomarbeit lag in der Aktualität der beschriebenen und verwendeten Architekturen und Technologien, welche einen Mangel an fundierter Literatur und stabilen Laufzeitumgebungen sowie einen stetigen Weiterentwicklungs- und Änderungsprozess implizierten. Teilweise musste, um die volle Funktionalität der Technologien auszunutzen, auf Alpha-Versionen zurückgegriffen werden, was die Arbeit nicht immer erleichterte, aber natürlich sehr interessant machte.

Zur Konzeption und Realisierung des Prototyps bediente sich der Autor einer Kombination aus dem ´Rational Unified Process´ und ausgewählten Elementen des ´Extreme Programming´. Diese Vorgehensweise kann, nach Meinung des Verfassers, für Projekte in diesem Rahmen sehr empfohlen werden, da sie sich insbesondere durch schnelle Reaktionsmöglichkeiten bei ständig wechselnden Anforderungen auszeichnet.

DANKSAGUNGEN

An dieser Stelle möchte ich mich bei meinen Betreuern für die geleistete Unterstützung bedanken, insbesondere bei Herrn Prof. Dr. Kaufmann, der in einigen Gesprächen und vielen E-Mails durch zahlreiche Anregungen zum Gelingen dieser Arbeit beigetragen hat.

Mein Dank geht auch an meine Kollegen, die durch anregende Diskussionen und konstruktive Kritik sowie aufmunternde Worte einen wichtigen Teil beitrugen, insbesondere an die, die Angst hatten an dieser Stelle vergessen zu werden, Herrn Dipl.-Ing. Peter Diel und Herrn Dipl.-Math. Gerhard Schneider.

Für ihre Unterstützung möchte ich mich bei meinen Eltern und Schwiegereltern bedanken.

Weiterhin bedanke ich mich bei meinen Kommilitonen und Freunden, die mir durch ihr großes Interesse an dieser Arbeit, durch Fragen und Diskussionen geholfen haben, dieses Gebiet in seiner Gesamtheit zu begreifen und aus verschiedenen Blickpunkten zu betrachten.

Für zahlreiche Tipps und Tricks bezüglich Design und Layout bedanke ich mich bei Sylvia Brandt.

Mein besonderer Dank geht an meine Verlobte Alexandra Georg für Ihr liebevolles Verständnis, ihre Unterstützung und ihre umfangreiche Hilfe beim Redigieren dieser Arbeit.

Darüber hinaus danke ich allen weiteren Personen, die auf irgendeine Art zum Gelingen dieser Arbeit beigetragen haben.

Februar 2002 Roger Zacharias

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Gleichzeitig versichere ich, diese Arbeit in gleicher oder ähnlicher Form weder veröffentlicht, noch einer anderen Prüfungsbehörde vorgelegt zu haben.

Dillenburg, den 15.02.2002

Inhaltsverzeichnis

1 EINFÜHRUNG 10

1.1 EINLEITUNG 10
1.2 ZIELSETZUNG 11
1.3 VORGEHENSWEISE UND AUFBAU DER ARBEIT 11
1.4 BESCHREIBUNGSMETHODEN 12

2 ARCHITEKTUREN, TECHNOLOGIEN UND PARADIGMEN 14

2.1 WEB SERVICE ARCHITEKTUREN 15
2.1.1 Ein neues Paradigma 15
2.1.2 XML (Extensible Markup Language) 29
2.1.3 SOAP (Simple Object Access Protocol) 44
2.1.4 WSDL (Web Services Description Language) 49
2.1.5 UDDI (Universal Description, Discovery, Integration) 52
2.1.6 Web Services Frameworks 55
2.1.7 Auswirkungen auf Geschäftsprozesse und –modelle 66
2.1.8 Zusammenfassung und Ausblick 67
2.2 MANAGEMENT-ARCHITEKTUREN 69
2.2.1 Begriffe, Anforderungen, Historie und Entwicklungen 71
2.2.2 Terminologie und grundlegende Konzepte 81
2.2.3 OSI-Management / CMIP 87
2.2.4 Internet-Management / SNMP 96
2.2.5 Vergleich OSI- und Internet-Management 101
2.2.6 JMX – Die Management-Architektur von Morgen? 103
2.2.7 Integrierte Managementsysteme 114
2.3 TECHNOLOGIEN ZUM ZUGRIFF AUF BANKPERIPHERIEGERÄTE / J/XFS 117
2.3.1 Historie 117
2.3.2 Technologie, Architektur und Funktionsweise 121
2.3.3 Zusammenfassung 128

3 EIN ÜBERWACHUNGSSYSTEM FÜR BANKPERIPHERIEGERÄTE 132

3.1 FACHLICHER HINTERGRUND 132
3.1.1 Management von und durch Web Services 133
3.2 ANFORDERUNGSANALYSE 137

3.2.1 Systemspezifikation 137
3.2.2 Nichtfunktionale Anforderungen 138
3.2.3 Funktionale Anforderungen 139
3.2.4 Akteure und Anwendungsfälle 140
3.3 KONZEPTIONELLES MODELL 150
3.3.1 Statik des Systems 150
3.3.2 Dynamik des Systems 159
3.4 GROBENTWURF (TOP LEVEL DESIGN) 161
3.4.1 Benutzerschnittstelle 161
3.4.2 Architektur 164
3.4.3 Technologien, Produkte und APIs 165
3.5 FEINENTWURF (DETAIL DESIGN) 175
3.5.1 Deployment 176
3.5.2 Managementdaten 179
3.5.3 Informationsfluss 180
3.5.4 Detaillierte Statik und Dynamik 183
3.6 IMPLEMENTIERUNG 184

4 BEWERTUNG 186

5 ZUSAMMENFASSUNG 192

KAPITEL **1**

Einführung

One does not discover new lands without consenting to lose sight of the shore for a very long time.
(Andrew Gide)

1 Einführung

1.1 Einleitung

Nichts ist beständiger als der Wandel! Diese Erkenntnis trifft momentan genau den Kern. In allen Bereichen der IT werden bestehende Architekturen, Produkte und Prozesse hinterfragt. Einhergehend mit dieser Entwicklung ist das Bestreben nach Standardisierung, Offenheit und die Nutzung bereits vorhandener und bewährter Implementierungen zu beobachten, ohne aber bereits getätigte Investitionen zu vernachlässigen. Im Bereich der Softwareentwicklungsprozesse konkurrieren RUP[1] und XP[2] (oder werden sinnvoll gemeinsam eingesetzt), UML[3] hat sich als Standard durchgesetzt (ist aber mittlerweile für viele schon zu umfangreich geworden), XML[4] ist kein ´Wunderding´ mehr, Begriffe wie ´Web Services´[5], ´SOAP´[6] und ´EAI´[7] kursieren (und werden überall anders definiert), mächtige Frameworks wie ´J2EE´[8] entstehen, worauf Microsoft mit ´.NET´[9] antwortet und SAP liegt mit ´mySAP.com´[10] voll im Trend. Das Internet ist unersetzbarer Bestandteil unseres Lebens geworden, der E-Commerce kommt langsam in Fahrt und es wächst der Wunsch der Unternehmen, Prozesse über das Internet mit anderen Unternehmen zu integrieren. Viele dieser Entwicklungen sind mit weitreichenden Marketing-Maßnahmen verbunden, die diesen zunächst den Status des Hype verleihen, sich aber wie XML später doch als ´ganz brauchbar´ herausstellen. Auch im Bereich der Banken- bzw. Überwachungs-Software, in dem der praktische Teil dieser Arbeit angesiedelt ist, gelten diese Regeln. Globalisierung, Vernetzung, anspruchsvollere Konsumenten und verstärkter Konkurrenzdruck sind die Herausforderungen an die Finanzwirtschaft, in der, wie in kaum einem anderen Bereich, die IT-Systeme durch

[1] Unter dem Rational Unified Process (RUP) versteht man ein objektorientiertes, UML-basiertes Vorgehensmodell, welches gleichzeitig kommerziell von der Fa. Rational Software vertrieben wird. Weitere Informationen unter http://www.rational.com/rup.

[2] Unter Extreme Programming (XP) versteht man einen leichtgewichtigen Entwicklungsprozess für kleine Teams, der aus 12 Richtlinien besteht und hierbei auch bestimmte soziale Aspekte des Software-Engineerings berücksichtigt. Weitere Informationen unter http://www.extremeprogramming.org.

[3] Die Unified Modelling Language (UML) ist eine grafische Notation zur Erstellung objektorientierter Modelle für die Analyse und den Entwurf objektorientierter Software. Weitere Informationen unter http://www.uml.org.

[4] Die Extensible Markup Language (XML) ist eine Metasprache zur Definition von Markup-Sprachen. Eine Einführung in XML bietet *Kapitel 2.1.2 – XML (Extensible Markup Language)*.

[5, 6, 7, 8, 9, 10] Auf diese Begriffe wird in *Kapitel 2 – Architekturen, Technologien und Paradigmen* näher eingegangen.

beträchtliche Kosten, enorme Komplexität und hohe Anforderungen an die Funktionalität gekennzeichnet sind.

Die Lösung sind moderne Architekturen, die leicht in bereits bestehende heterogene Strukturen integrierbar, beliebig erweiterbar und adaptierbar sind und dabei im Idealfall auf offenen Standards basieren. Web Services versprechen diese Anforderungen zu erfüllen und sollen die nächste Generation des Internets - das ´service web´ - ermöglichen. Gleichzeitig steigt der Bedarf an Managementsystemen, welche in der Lage sind diese komplexen Systeme, welche manuell nicht mehr beherrschbar sind, zu überwachen und zu administrieren. Auch hier existieren eine Reihe neuer Technologien und Architekturen, die dies leisten wollen.

1.2 Zielsetzung

Diese Diplomarbeit verfolgt zwei Ziele: Zum einen soll sie in die Gebiete der Management- und Web Services-Architekturen einführen, deren Vor- und Nachteile, Einsatzgebiete und Entwicklung diskutieren und sie, sowie entsprechende ´state of the art´-Lösungen, damit dem Leser näher bringen. Zum anderen soll sie, basierend auf diesen Erkenntnissen, durch die prototypische Konzeption und Realisierung eines Überwachungssystem für Bankperipheriegeräte die enormen Möglichkeiten und Potentiale aufzeigen, wie durch die Kombination von Management- und Web Services-Architekturen lose-gekoppelte, plattformunabhängige, leicht integrierbare und adaptierbare Management-Systeme erstellt werden können – ein Aspekt, der nach Wissen des Autors zur Zeit noch nicht in der Literatur zu finden ist. Die Funktionalität des erstellten Prototyps umfasst nur die absolut notwendigen Aufgaben von Management-Systemen. Dieser Prototyp kann und wird entsprechend erweitert werden, um in eine zurzeit bei der Wincor Nixdorf GmbH & Co. KG entwickelte moderne Multichannel-Architektur einzufließen.

1.3 Vorgehensweise und Aufbau der Arbeit

Kapitel 2 – Architekturen, Technologien und Paradigmen – führt in das Umfeld der Management- und Web Services-Architekturen sowie in die für die Konzeption und Realisierung des Überwachungssystems notwendigen Technologien im Banken-Umfeld ein.

Kapitel 3 – Ein Überwachungssystem für Bankperipheriegeräte – beschreibt die Konzeption und Realisierung eines Überwachungssystems für Bankperipheriegeräte unter Verwendung der im zweiten Kapitel beschriebenen Konzepte.

Kapitel 4 – Bewertung – beurteilt die Ergebnisse, zu denen der Autor durch seine wissenschaftliche Auseinandersetzung mit diesen Themen gekommen ist.

Kapitel 5 – Zusammenfassung – fasst die wichtigsten Punkte dieser Arbeit zusammen und geht noch einmal auf die enormen Möglichkeiten ein, die sich aus der Verwendung dieser Architekturen bzw. Technologien ergeben.

1.4 Beschreibungsmethoden

Zur Darstellung der entwickelten Konzepte wird in dieser Arbeit, neben der verbalen Beschreibung, auch mit der zur Zeit am weitesten verbreiteten grafischen Notationssprache, der Unified Modelling Language (UML), gearbeitet. Die UML ist die Vereinigung und Weiterentwicklung verschiedener Analyse- und Design-Notationen, wurde 1997 von der OMG[1] als Notation für Softwarearchitekturen standardisiert und liegt zum Zeitpunkt dieser Arbeit in der Version 1.4 vor.

[1] Die Object Management Group (OMG) ist eine internationale, nicht profit-orientierte, herstellerübergreifende Organisation, die von einigen großen IT-Firmen mit dem Ziel gegründet wurde, die Entwicklung von objektorientierter Software und Technologie zu unterstützen. Dies geschieht hauptsächlich durch die Erstellung von Richtlinien und Standards.

KAPITEL **2**

Architekturen, Technologien und Paradigmen

Computers are useless – they can only give you answers.
(Pablo Picasso)

2 Architekturen, Technologien und Paradigmen

Dieses Kapitel soll in die Web Services-, Management- und Bankperipheriezugriffs-Architekturen und Technologien einführen und damit verknüpfte Paradigmen aufzeigen. Das Verständnis dieser Punkte ist Voraussetzung für die in *Kapitel 3 – Ein Überwachungssystem für Bankperipheriegeräte* vorgestellte Konzeption und Realisierung eines Überwachungssystem-Prototyps. Dieser baut auf diesen Technologien auf und versucht ihre Vorteile zu nutzen, indem er sie sinnvoll verknüpft und anwendet. Da sowohl XML als auch SNMP mittlerweile weit verbreitetet und bekannt sind, können Leser, die in diesen Bereichen entsprechende Kenntnisse besitzen, *Kapitel 2.1.2 – XML (Extensible Markup Language)* und *Kapitel 2.2.4 – Internet Management / SNMP* ohne Beeinträchtigung auslassen. Die Aufteilung dieses Kapitels stellt sich folgendermaßen dar:

- Web Services Architekturen
- Management Architekturen
- Architekturen zum Zugriff auf Bankperipheriegeräte

Zu jedem Themengebiet werden zunächst Historie und Notwendigkeit beschrieben, anschließend Begrifflichkeiten geklärt und tiefer in die Technologie eingestiegen. Am Ende jedes Themengebiets werden verschiedene Anwendungsbeispiele dargestellt, wie diese Architekturen heute in der IT verwendet werden. Aufgrund der Komplexität jeder dieser drei Bereiche soll jeweils nur auf die wichtigsten übergreifenden Punkte und die für den Prototyp notwendigen Details eingegangen werden. Für denjenigen Leser, welcher tiefer in die jeweilige Thematik einsteigen möchte, bietet das Literaturverzeichnis eine Reihe weiterführender Literatur.

2.1 Web Service Architekturen

Firma X möchte im Internet die Ware W bei Firma Y bestellen, wobei Y diejenige Firma sein soll, welche W am kostengünstigsten anbietet. Um den günstigsten Anbieter zu finden, muss ein Mitarbeiter der Firma X mit Hilfe von Suchmaschinen und Katalogen verschiedene Anbieter auswählen und auf deren Webseiten nach den Konditionen recherchieren. Günstiger wäre es, die Anforderungsparameter, also Produkt, Kosten, Lieferbarkeit, Zahlungsbedingungen usw. über eine geeignete Schnittstelle zu definieren und damit einen Software-Agenten zu veranlassen, alle Angebote zu durchsuchen um den optimalen Anbieter zu finden. An dieser Stelle setzt die Vision des dynamischen E-Business unter Nutzung der ´Web Services´ Technologie an. In diesem Kapitel sollen Web Services, ihre Visionen, die zugrunde liegenden Technologien sowie Auswirkung auf E-Business und Geschäftsmodelle diskutiert werden.

2.1.1 Ein neues Paradigma

Das Problem ist bekannt: Es existieren Millionen von Web-Servern weltweit, welche fast das gesamte Wissen der Menschheit jedermann zugänglich machen sollen. Die übliche Form, in der Informationen geliefert werden, ist HTML, welches – von einem Browser dargestellt – den Inhalt dem menschlichen Betrachter präsentiert. Das heißt, das World Wide Web ist zurzeit fast ausschließlich für die Maschine-Mensch-Kommunikation vorgesehen (siehe hierzu auch *Kapitel 2.1.2.1 – Warum XML?*). Web Services sind derzeit eines der am heißesten diskutierten Themen, versprechen sie doch, die Informationen auch Maschinen (Software) zugänglich zu machen, also zusätzlich die Möglichkeit der ´program-to-program interaction´ zu eröffnen. Gerade diese Möglichkeit, das volle Potential des Internets zu nutzen, ist heute für ausgefeilte B2B-, B2C- oder P2P-Lösungen von zunehmender Bedeutung.

Mit CORBA[1], RMI[2], DCOM[3] usw. existieren bereits verschiedene Protokolle für den Zugriff auf entfernte Dienste. Diese sind aber komplex, teuer und meist nicht interoperabel. Weiterhin eignen sich diese Protokolle aus dem CORBA-, Java- und

[1] Bei der Common Object Request Broker Architecture (CORBA) handelt es sich um die zurzeit bekannteste und ausgereifteste Middleware-Architektur. Diese Plattform ist sowohl programmiersprachen- als auch plattformunabhängig.

[2] RMI (Remote Method Invocation) ist das verteilte Komponentenmodell der Java-Plattform.

[3] Die Distributed Components (DCOM) Architektur ist das verteilte Komponentenmodell der Microsoft-Plattform.

Microsoft-Bereich nicht für das Internet, da sie verbindungsorientiert sind und stabile Verbindungen erfordern, die das Internet nicht bieten kann. Außerdem sind diese Protokolle für Aufrufe über das Internet ungeeignet, da sie aufgrund variabler Portzuordnungen und ähnlichem Probleme mit dem Verbindungsaufbau durch Firewalls haben. Diese ´tightly coupled systems´ werden durch ´message oriented middleware´ (MOM) ergänzt, welche die zeitliche Entkoppelung von Systemen durch asynchrone Nachrichtenverarbeitung ermöglicht, aber im Internet ähnliche Probleme wie die oben genannten besitzt. Das grundsätzliche Problem besteht darin, dass Middleware-Technologien und das Internet zwei völlig getrennte Welten darstellen. Die Idee besteht nun darin, diese beiden Welten zusammenzubringen und das Internet zu einer Art Middleware zu erweitern. Man setzt für die Kommunikation das übliche Web-Protokoll HTTP ein und benutzt für den Datentransport XML, welches selbstbeschreibend und interoperabel ist. Als Messaging-Protokoll wird die im Internet bereits existierende Messaging-Lösung eingesetzt: E-Mail über SMTP. Dieses Szenario bzw. diese Vision wird als ´Web Service Technologie´ bezeichnet.

Die Web Service Technologie ist also eher Evolution als Revolution. Bereits existierende Technologien bilden die Grundlage, auf denen Web Services aufbauen. Oder um es mit James Harts[1] Worten zu sagen: "Web Services are, in many ways, the logical conclusion of several threads which have been developing in the IT industry for twenty years. However, this doesn't mean we should underestimate their significance. By combining the steam engine and the wheel, George Stephenson was simply combining the logical conclusions of two established fields of engineering into their logical conclusion, but the implications of the invention of railroads can still be felt today. Web Services have something of the feel of that first steam train about them – it's going to be an interesting ride."[2] Folgende Abbildung verdeutlicht die Entwicklungen, welche die Basis für Web Services legten.

[1] James Hart ist einer der Autoren des bekannten Buches ´Professional Java XML´, Wrox Press.

[2] Vgl. [JHWS01], S. 2.

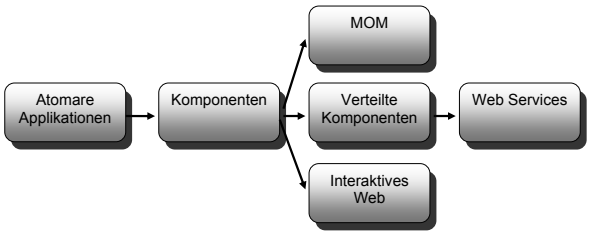

Abbildung 2-1 Entwicklungsgeschichte der Web Services

Hierbei ist jedoch zu beachten, dass Web Services als Ergänzung zu den bereits existierenden Technologien betrachtet werden müssen. Die Web Service Technologie dient dazu, Geschäftslogik über das Internet zur Verfügung zu stellen, wobei Technologien wie J2EE und CORBA der Implementierung dieser Geschäftslogik dienen.

Da Web Services von allen Global Players der IT, d.h. IBM, Microsoft, Sun, HP, Oracle usw. propagiert werden, existieren für den Begriff ´Web Service´ eine Reihe von Definitionen. Eine, nach Ansicht des Autors, geeignete findet sich in [CBDIWS]: "A web service provides programmable access to a capability that can be used independently of its implementation via a self-describing interface based on open internet standards." Ein Web Service stellt eine Fähigkeit (capability) zur Verfügung – dies ist der geleistete Dienst (´service´). Der Zugriff auf diesen Dienst erfolgt via Internet-Protokollen – daher der Begriff ´web´. Bei diesen Protokollen handelt es sich um ´offene Standards´, d.h. Standards, auf welche sich die Mehrzahl der Unternehmen geeinigt haben und die vom W3C verwaltet werden. Weiterhin können Web Services dynamisch gefunden und aufgrund des ´programmierbaren Zugriffs´ in eigene Applikationen integriert werden.

2.1.1.1 Web Service Perspektiven

Ein Web Service kann aus zwei Perspektiven betrachtet werden: aus der Sicht des Konsumenten des Dienstes und aus der Sicht des Erbringers (Provider) des Dienstes. Für den Konsumenten ist nur der erbrachte Dienst entscheidend, d.h. es interessiert, welche Aufgabe der jeweilige Dienst besitzt, wie er benutzt werden kann

und auffindbar ist. Entscheidend für den Konsumenten ist also nur die Dienstbeschreibungsschnittstelle, so dass aus dieser Sicht ein Web Service ein Dienst ist, welcher über eine Schnittstelle im Internet angeboten wird. Für den Provider ist neben der Schnittstelle die Implementierung des Dienstes bestimmend. Der Provider entscheidet, welche Funktionalität der Dienst bietet, an welcher Stelle er sich zur Laufzeit befindet und wo sein Vorhandensein publiziert ist. Aus der Sicht des Providers handelt es sich also bei einem Web Service sowohl um die offen gelegte Schnittstelle als auch die Software, welche den Dienst implementiert. Die folgende Abbildung verdeutlicht dies.

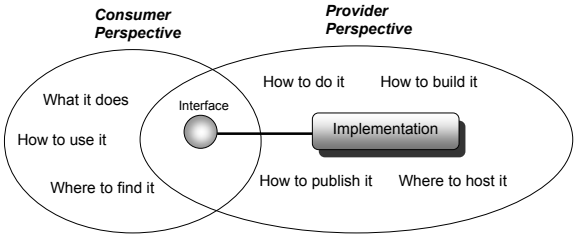

Abbildung 2-2 Web Service Perspektiven

2.1.1.2 Web-Applikations Topologien

Was unterscheidet die Web Service Technologie von anderen Topologien, wie zum Beispiel ´Application Service Providing´ (ASP) oder ´Software on Demand´ (SOD)? Application Service Provider stellen meist Komplettlösungen einer Software zur Verfügung, d.h. die gesamte Applikation im Sinne von Benutzeroberfläche, Workflow, Geschäfts- und Daten-Komponenten ist in einer Lösung gebunden. Diese Komplettlösung wird auf den Servern der ASPs gelagert, verwaltet und ausgeführt. Der Benutzer dieser Software greift meist mittels Browser auf sie zu, um damit zu arbeiten. Es existieren in den meisten Fällen wenige Möglichkeiten diese Software zu konfigurieren und sie in die eigenen Geschäftsprozesse einzubinden. Eine Alternative bietet die Software on Demand Lösung, bei welcher die Software auf Bedarf (on demand) auf den Rechner des Kunden geladen wird. Diese Software kann am Ende der Sitzung gelöscht werden oder auf dem Rechner verweilen, bis sie durch eine neue Version ersetzt wird oder ein bestehender Nutzungsvertrag ausläuft.

Diese Lösung bietet die Möglichkeit der unbeschränkten Konfiguration und Personalisierung. Web Services bieten eine weit flexiblere Lösung: Der Kern der Applikation, d.h. Geschäfts- und Datenkomponenten lagern weiterhin auf dem Server, es kann nun aber programmatisch über ihre Schnittstellen auf sie zugegriffen werden. So kann der Kunde seine eigenen Geschäftsprozesse und Benutzeroberflächen erstellen und in diese beliebige Web Services beliebiger Provider einbinden. Folgende Abbildung verdeutlicht dies.

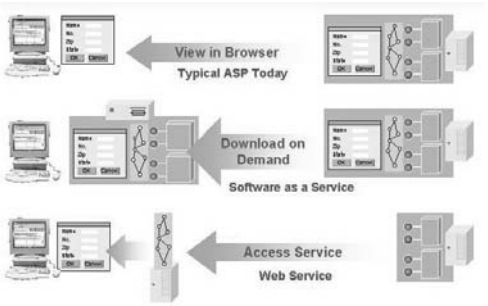

Abbildung 2-3 ASP, SOD und Web Services (Quelle: [CBDIWS], S.11)

2.1.1.3 Zukunfts-Szenario

Web Services können, wie bereits angesprochen, in allen Bereichen verwendet werden, wobei ihr größtes Potential sicher in den Bereichen B2B und B2C liegt. Ein E-Commerce-Szenario der Zukunft, bei welchem die Informationen der Wertschöpfungskette eines Unternehmens ausschließlich über Web Services ausgetauscht werden, wird in *Abbildung 2-4 E-Commerce Szenario der Zukunft* dargestellt. Die Vorteile stellen sich folgendermaßen dar: Zunächst ist es möglich, direkt mit der jeweiligen Informationsquelle zu kommunizieren, wodurch jedes Datum der Wertschöpfungskette immer akkurat und aktuell ist und damit die Notwendigkeit der regelmäßigen Replikation entfällt. Verlangt der Kunde Informationen zur Lieferbarkeit eines Produktes, wird diese Anfrage automatisch an den entsprechenden Lieferanten weitergeleitet und von diesem beantwortet. Auf gleichem Weg wird das Problem der ungenauen, falschen oder veralteten Daten behoben. So

ist es sowohl für den Kunden als auch für das Unternehmen möglich, bestimmte Web Services der Wertschöpfungskette bzw. des Kunden-Frontends unabhängig von Plattform und Endgerät in die eigene Applikation oder Kette einzubinden.

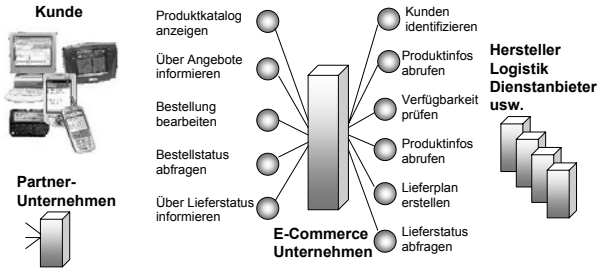

Abbildung 2-4 E-Commerce Szenario der Zukunft

2.1.1.4 Web Services Modell

Die Web Services Architektur basiert auf Interaktionen zwischen drei Instanzen, welche jeweils eine bestimmte Rolle einnehmen. Diese Instanzen sind:

- Service Provider
  Der Service Provider stellt den Web Service zur Verfügung, welcher einen spezifischen Dienst leistet. Aus der Geschäftsperspektive ist dies der Besitzer des Dienstes, aus der Architekturperspektive derjenige, welcher den Dienst hostet und den Zugriff auf diesen ermöglicht.
- Service Requestor (Consumer)
  Der Service Requestor ist der Konsument des Dienstes. Er findet und benutzt den Dienst, um eine bestimmte Aufgabe zu erfüllen. Aus der Geschäftsperspektive ist dies das Unternehmen, welches eine bestimmte Funktionalität benötigt, aus der Architekturperspektive ist dies die Applikation, welche nach einem Dienst sucht und eine Interaktion mit diesem Dienst anstrebt. Die Rolle des Service Requestors kann hierbei von einem Browser, welcher auf Benutzereingaben reagiert, oder einem Programm ohne Benutzerschnittstelle gespielt werden. Natürlich kann der Service Requestor selbst ein Web Service sein, wodurch Workflows realisiert werden können.

- Service Registry (Broker)
  Die Service Registry bietet ein durchsuchbares Verzeichnis für Web Services Schnittstellen. Service Requestors können in diesem Verzeichnis Dienste auffinden, um diese zu nutzen.

Um das volle Potential dieser Architektur auszunutzen, müssen drei Interaktionen stattfinden, welche sowohl einzeln als auch iterativ auftreten können:

- publish
  Damit auf einen Web Service zugegriffen werden kann und der Service Requestor diesen finden kann, muss dessen Schnittstelle veröffentlicht werden. An welcher Stelle die Schnittstelle veröffentlicht wird, hängt von verschiedenen Faktoren ab. Ist eine dynamische Bindung gewünscht, geschieht dies in der Service Registry. Andernfalls kann die Schnittstellenbeschreibung auch direkt zwischen Service Provider und Service Requestor ausgetauscht werden (zum Beispiel über E-Mail, FTP, Veröffentlichung auf Website usw.).
- find
  Mit Hilfe dieser Operation erlangt der Service Requestor die Schnittstellenbeschreibung des gewünschten Dienstes. Dies kann direkt oder über das Suchen innerhalb der Service Registry geschehen. Weiterhin kann diese Operation in zwei verschiedenen Lebenszyklusphasen aufgerufen werden: zur Designzeit, um die Schnittstellenbeschreibung zur Programmentwicklung zu verwenden und zur Laufzeit, um eine dynamische Nutzung der Architektur zu ermöglichen.
- bind
  Durch diese Operation baut der Service Requestor eine Verbindung zum Service Provider auf und initiiert eine Interaktion. Das Initiieren der Interaktion geschieht entweder durch einen direkten Aufruf einer Methode des Web Service (RPC) oder dadurch, dass sich der Service Requestor für asynchrone Ereignisse beim Service Provider registriert (Messaging).

Wie bereits in *Kapitel 2.1.1.1 – Web Service Perspektiven* angesprochen, hat ein Web Service zwei grundlegende Bestandteile:

- Service
  Die Implementierung des Web Service liefert den Service selbst. Dieser Service ist ein Softwaremodul, welches auf dem Server des Service Providers abgelegt ist

und per Netzwerk zugreifbar ist. Es interagiert mit dem Service Requestor oder kann selbst als Service Requestor fungieren, indem es andere Web Services benutzt.

- Service Description
  Die Beschreibung eines Web Service beinhaltet die Details der Schnittstelle und der Implementierung des Dienstes. Dies umfasst Datentypen, Operationen, Bindungsinformationen und Informationen über den Server, auf welchem sich der Dienst befindet. Es können weiterhin Metainformationen über den Dienst und dessen Provider enthalten sein, um den Web Service zu kategorisieren und das Auffinden durch den Service Requestor zu vereinfachen.

Das typische Szenario sieht folgendermaßen aus: Ein Service Provider stellt unterschiedliche Dienste, wie zum Beispiel einen Aktienkurs-Informationsdienst, auf seinem Server zur Verfügung. Desweiteren veröffentlicht er die Web Service Beschreibung in einer globalen Service Registry, wo diese von jedem aufgefunden werden kann. Ein Service Requestor sucht in dieser Registry nach für ihn geeigneten Diensten und findet zum Beispiel diesen Aktienkurs-Informationsdienst. Er erhält als Antwort auf seine Anfrage an die Registry die Schnittstellenbeschreibung des Dienstes und ist nun in der Lage auf diesen zuzugreifen, um ihn zum Beispiel in eine eigene Applikation einzubinden. Das Web Services Modell stellt sich also folgendermaßen dar:

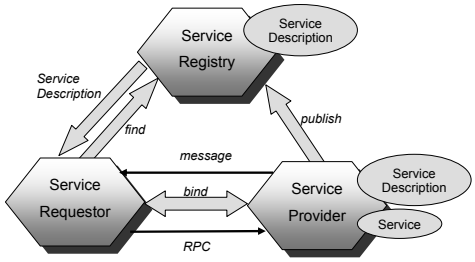

Abbildung 2-5 Web Services Modell

2.1.1.5 Warum Web Services?

Es existiert eine Reihe von Vorteilen, welche sich durch die Nutzung von Web Services ergeben. Einige dieser Vorteile können auch durch andere Ansätze und bereits existierende Technologien erreicht werden. Nachfolgend verschiedene für die Web Service Technologie spezifische Vorteile:

- Loose technologie coupling – zwischen Requestor- und Provider-Plattform besteht keine Technologieabhängigkeit mehr.
- Late binding – ermöglicht den Beziehungsaufbau zwischen Systemen zur Laufzeit.
- Dynamic Inspection – die Fähigkeiten und Funktionalitäten von Web Services können auch zur Laufzeit aufgefunden werden.
- Programmable – Web Services ermöglichen es, dass Systeme Funktionen über das Internet aufrufen, statt nur Daten auszutauschen und zu präsentieren.
- Use of standard protocols – es wird eine Reihe standardisierter und allgemein anerkannter Protokolle verwendet, statt wie bisher viele verschiedene proprietäre Technologien.
- Industry consensus – alle Key Player akzeptieren und propagieren die Web Service Technologie.
- Protecting investments – Web Services ersetzen keine bestehenden Systeme, sondern fügen eine Schicht weiterer Funktionalität hinzu. Dadurch sind bereits getätigte Investionen geschützt.

2.1.1.6 Web Service Standards

Web Services sind in jeder Netzwerk-Umgebung verwendbar, egal ob es sich dabei um Internet, Extranet oder Intranet handelt. Ihr alleiniger Fokus liegt auf dem Zusammenspiel der Anwendungskomponenten. Weiterhin spielen weder Betriebssystem noch Programmiersprache oder Komponentenmodell eine Rolle. Es liegt nahe, dass eine solche Offenheit der Zusammenarbeit einige Standards voraussetzt. Diese Standards, welche in den nachfolgenden Kapiteln näher vorgestellt werden sollen, sind: XML, SOAP, WSDL und UDDI. Es ist jedoch zu beachten, dass diese vier Standards nur die erste Stufe und damit die Basis der Web Services Evolution darstellen. Um die vollständige Vision zu verwirklichen, bedarf es

einer Reihe weiterer Standards, welche sich zum Teil bereits in der Entwicklung befinden. Der Ansatz, welcher insbesondere von IBM propagiert wird, besteht in einer modularen Annäherung an die Gesamtvision, d.h. die Evolution verläuft in Phasen. Dies impliziert, dass jedes Protokoll separat eingeführt wird und Zeit hat sich selbständig zu entwickeln.

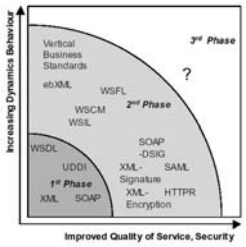

Abbildung 2-6 Web Services Evolution (Quelle: [WSRHRN],S.5)

Eine Erklärung der dargestellten Technologie liefert die folgende Tabelle:

XML	Extensible Markup Language; Metasprache zur Definition von Markup-Sprachen
SOAP	Simple Object Access Protocol; XML-Protokoll zum Transport von XML-Daten
WSDL	Web Services Description Language; XML-Protokoll zur Beschreibung von Web Services
UDDI	Universal Description, Discovery, Integration; Basis für Web Services Registry
WSIL	Web Services Inspection Language; Durchsuchen einer Website nach verfügbaren Web Services
WSCM	Web Services Component Model; Komposition und Präsentation von Web Services
ebXML	Electronic Business Extensible Markup Language; XML-basierte Infrastruktur für den elektronischen Handel
XML-Signature	Beschreiben und Prüfen von digitalen Signaturen auf XML-Basis

XML-Encryption	Verschlüsselung von digitalen Inhalten auf XML-Basis
SOAP-DSIG	Simple Object Access Protocol – Digital Signatures; Signieren von SOAP Nachrichten
SAML	Security Assertion Markup Language; Austausch von Authentifikations- und Autorisierungsinformationen
HTTPR	Hypertext Transfer Protocol Reliable; Zuverlässiger Datentransport über HTTP
WSFL	Web Services Flow Language; Workflow auf Web Services Basis

Tabelle 2-1 Web Services Standards

Wir befinden uns zurzeit in der ersten Phase (2001 – dynamic business integration, simple transactions). Die Basisprotokolle (XML, SOAP, WSDL und UDDI) sind standardisiert oder zumindest de facto Standards. In dieser Phase sind bereits folgende Szenarien möglich:

- Verbesserte Enterprise Application Integration (EAI) – Web Services werden intern zur vereinfachten Anbindung an Legacy-Systeme verwendet.
- Business to Business (B2B) Integration zwischen Partnern – Web Services ermöglichen Echtzeit-Kommunikation zwischen ausgewählten Organisationen, unabhängig von der für die Implementierung verwendeten Technologien.
- Erstellung von intelligenten Endpunkten – Web Services liefern strukturierte Informationen zur externen Weiterverarbeitung in fremden Applikationen ohne die bisherige Abhängigkeit von Benutzerinteraktionen.
- Einfache Informationsdienste und Transaktionen sind für eine weitreichendes Publikum verfügbar.

Obwohl die Web Service Technologie noch in ihren Kinderschuhen steckt, wird sie von allen Seiten akzeptiert und durch eine Vielzahl von Entwicklungsumgebungen, Adaptern und Konnektoren unterstützt. "It is difficult to find a previous technology that the whole industry has been so quick to endorse and adopt without major dissent."[1]

[1] Vgl. [WSRHRN], S.5.

Phase zwei (2002-2004 – improved infrastructure, more complex transactions) wird weitere Protokolle einführen, welche eine robustere Infrastruktur und komplexere Transaktionsszenarios ermöglichen. Eine weitere wichtige Aufgabe dieser Protokolle ist die Schaffung einer Sicherheitsarchitektur für Web Services.

In Phase drei (2004+ – adaptive systems) haben Web Services eine weite Verbreitung gefunden und Organisationen werden nun nicht nur ihre Systeme, sondern auch ihre Geschäftsmodelle anpassen, um die Vorteile des dynamischen Verhaltens der Web Service Technologie zu nutzen. Dazu gehören das dynamische Auffinden von Geschäftspartnern, die dynamische Nutzung von entfernten Diensten und die dynamische Anpassung an Änderungen aller Art. Es werden sich selbst beschreibende und sich selbst heilende Systeme möglich, welche sich permanent optimieren, um neue Möglichkeiten zu nutzen.

2.1.1.7 Web Services Stack

Um die Operationen publish, find und bind interoperabel auszuführen, wird ein Web Services Stack benötigt, welcher auf jeder Ebene entsprechende Standards und Protokolle zur Verfügung stellt. *Abbildung 2-7 Web Services Stack* zeigt diesen Web Services Stack. Die jeweils über einer anderen liegende Schicht nutzt die Fähigkeiten der darunter liegenden Schichten und baut auf diesen auf. Die vertikalen Balken repräsentieren Anforderungen, welche in jeder Schicht erfüllt sein müssen. Der Text auf der linken Seite nennt die Standards bzw. aufkommende Standards, die die jeweilige Schicht adressieren.

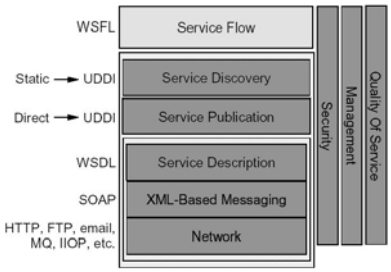

Abbildung 2-7 Web Services Stack (Quelle: [WSCA], S.10)

Die Grundlage des Web Services Stacks bildet das Netzwerk. Web Services müssen über das Netzwerk zugreifbar sein, um den Informationsaustausch zwischen Service Requestor und Service Provider zu ermöglichen. Web Services im Internet-Bereich nutzen weit verbreitete Netzwerkprotokolle wie HTTP, SMTP und FTP, während Web Services im Intranet Bereich weitere Protokolle wie IIOP[1] oder MQ (Message Queuing) benutzen können. Die zweite Schicht, das XML-basierte Messaging, repräsentiert die Benutzung von XML (siehe *Kapitel 2.1.2 – XML (Extensible Markup Language)*) als Basis für ein Messaging-Protokoll. In diesem Bereich hat sich SOAP (siehe *Kapitel 2.1.3 – SOAP (Simple Object Access Protocol)*) aus den verschiedensten Gründen als alleiniger Standard durchgesetzt. Die Service Description Schicht besteht eigentlich selbst wiederum aus einem Stack von Web Service Beschreibungs-Dokumenten, wobei WSDL (siehe *Kapitel 2.1.4 WSDL (Web Service Description Language)*) der minimale de facto Standard ist, um Web Services zu beschreiben. Weitere Beschreibungen auf höherer Ebene, wie zum Beispiel den Unternehmenskontext, Servicegüte oder Dienst-Dienst-Beziehungen bieten andere Protokolle. Der Unternehmenskontext wird zum Beispiel durch UDDI (siehe *Kapitel 2.1.5 – UDDI (Universal Description, Discovery, Integration)*) beschrieben, welches in Verbindung mit WSDL eingesetzt wird. Service Komposition und Workflow werden in einem WSFL Dokument beschrieben. Jede Aktion, welche ein WSDL Dokument einem Service Requestor zugänglich macht, wird als ´Service Publication´ bezeichnet. Dabei besteht die einfachste und zugleich am wenigsten dynamische Möglichkeit darin, das WSDL Dokument direkt an den Service Requestor zu senden. Alternativ kann der Service Provider das Dokument in einer öffentlich zugänglichen UDDI-Registry veröffentlichen, wo es für den Service Requestor auffindbar ist. Das Auffinden des Dokumentes geschieht ebenfalls mit Hilfe des UDDI Protokolls und wird als ´Service Discovery´ bezeichnet.

Da die Implementierung eines Web Services ein Softwaremodul ist, liegt es nahe, Web Services durch Komposition anderer Web Services zu erzeugen. Hierbei existieren eine Reihe von Anwendungsgebieten. So könnten unternehmensinterne Web Services kollaborieren, um Workflows abzubilden, wobei nach außen hin nur ein Web Service sichtbar ist, welcher die Workflow-Kette anstößt. Eine weitere wichtige Anwendung ist die Kollaboration von Web Services verschiedener Unternehmen zum

[1] IIOP (Internet Inter ORB Protocol) ist das CORBA-Protokoll für die Kommunikation im Internet.

Aufbau von Supply Chain Ketten. Einige Anwendungen werden in *Kapitel 2.1.7 – Auswirkungen auf Geschäftsprozesse und -modelle* vorgestellt. Die oberste Schicht, die Service Flow Schicht, beschreibt Service-Service-Kommunikation, Workflows und Kollaborationen. Der Standard, welcher sich zurzeit für diesen Bereich etabliert, ist WSFL.

Web Service Anwendungen, die heutigen E-Business Anforderungen genügen wollen, verlangen nach einer entsprechenden Infrastruktur, die den Anforderungen an Sicherheit, Management und Dienstgüte genügen. Diese vertikalen Balken adressieren jede Schicht und befinden sich zurzeit größtenteils noch in der Entwicklung.

2.1.2 XML (Extensible Markup Language)

Dieses Kapitel soll Grundlagen und Konzepte der Metasprache XML vorstellen, wozu zunächst kurz auf die Geschichte der Technologie eingegangen wird. Anschließend wird XML in das Umfeld der Meta- und Markup-Sprachen positioniert, die Vorteile dargestellt und diskutiert, warum XML im Begriff ist, die vorherrschende Sprache für Datenbeschreibung, Datenspeicherung und Datentransport zu werden.

2.1.2.1 Warum XML?

Die rasante Verbreitung des Internets, welches hauptsächlich auf dem Transportprotokoll HTTP (Hypertext Transfer Protocol) und der Markup-Sprache HTML (Hypertext Markup Language) basiert, brachte eine Menge Nachteile und Unzulänglichkeiten mit sich:

- Die Menge an unstrukturierten Informationen ist so gigantisch, dass es sehr schwer ist, die relevanten Informationen zu finden. Simon Phipps, CTO Sun Microsystems Inc., meinte zu diesem Problem: “The web could be a much better business tool if its data was more structured!“.
- Ein weiteres Problem liegt in der unterschiedlichen Darstellung der Informationen. Es existiert kein absoluter Standard für die Speicherung von Daten und die Kommunikation. So durchläuft ein Datum aus der Datenbank viele unterschiedliche Umformatierungen und Konvertierungen, um zum Beispiel auf einem WAP-Handy angezeigt zu werden oder in einer PDF-Datei zur Verfügung zu stehen.
- Die Inhalte unterliegen einer hohen Änderungsrate, da die Aktualität ein wichtiges Kriterium der dargestellten Informationen ist. Diese Aktualität kann aufgrund der unstrukturierten Informationsmenge nicht oder nur sehr schwer gewährleistet werden.
- Mit Hilfe der Darstellung und des Kontextes kann ein Mensch die Semantik der Daten erkennen, eine Maschine (Software) ist jedoch nicht in der Lage, semantische Informationen zu erkennen und zu extrahieren.
- Das WWW war ursprünglich für die Mensch-Mensch- bzw. Mensch-Maschine-Kommunikation konzipiert. Durch die Verwendung dieses Mediums als B2B-Plattform wird die Maschine-Maschine-Kommunikation immer wichtiger, was

vollkommen andere Anforderungen an die verwendeten Kommunikationssprachen stellt.

XML verfolgt hierbei vier grundlegende Ansätze:

- die Strukturierung der zurzeit noch unstrukturierten Informationen im WWW
- den plattform- und systemübergreifenden Datenaustausch zwischen Geschäftspartnern und zwischen Firmenentitäten innerhalb einer Firma
- die Beschreibung von Metadaten
- die Möglichkeit der maschinellen Verarbeitung der Daten

Mit XML war zunächst ein enormer Hype verbunden, mittlerweile hat sich aber herausgestellt, dass XML wirklich viele der beschriebenen Probleme lösen kann und damit seine Daseinsberechtigung in der IT-Welt hat. Die meisten Konzepte, die in XML verwendet werden, sind nicht neu. Wirklich neu ist die breite Unterstützung dieses Standards durch die Software-Industrie. Alle Global Player und Rivalen wie Sun, Microsoft, IBM, Oracle, HP, SAP usw. haben sich in Bezug auf diesen herstellerunabhängigen, ´freien´ Standard geeinigt. Viele Hersteller haben mittlerweile ihre proprietären Datenformate aufgegeben und nutzen XML. Die XML Technologie ist der Hoffnungsträger der IT-Welt: ein plattform- und applikationsübergreifendes Datenaustausch-, Datenverwaltungs- und Datendarstellungsformat, welches die divergierenden IT-Infrastrukturen weiter zusammenbringen und dem WWW zu mehr Struktur und Ordnung verhelfen soll. Mit Java ist erwiesenermaßen portabler Code möglich (´write once – run anywhere´), mit XML nun auch portable Daten (´write once – read anywhere´). Ein weiteres wichtiges Merkmal ist die strikte Trennung des Inhaltes von dessen Präsentation, d.h. in einem XML-Dokument werden Aussagen darüber gemacht, um welche Informationen es sich handelt und wie diese strukturiert sind, aber nicht, wie diese darzustellen sind. Hierzu existiert der XML Co-Standard XSL, auf welchen in *Kapitel 2.1.2.3.7 XSL – Transformation und Darstellung* kurz eingegangen werden soll. Weiterhin erlaubt XML den Anwendern ihre eigene, auf den entsprechenden Anwendungsbereich abgestimmte, Sprache zu definieren. Wichtig ist hierbei, dass die Dokumentenstruktur jeder mit XML erzeugten Sprache gleich bleibt und somit weiterhin unabhängig verarbeitet werden kann. Außerdem ist XML für Computer einfach zu verarbeiten, auszutauschen und darzustellen. Des Weiteren sind sehr viele Entwicklungswerkzeuge in diesem Umfeld frei erhältlich und vor allem

standardkonform. Diese Faktoren führten und führen zu einer weiten Verbreitung dieser Meta-Sprache.

2.1.2.2 Positionierung

Oft wird XML direkt mit HTML verglichen oder gar als dessen Nachfolger bezeichnet. Um eine klare Trennung zu erreichen, sollen zunächst die Begriffe der Meta- und Markup-Sprache, sowie des Dokuments geklärt werden.

Eine Metasprache dient zur Definition der Grammatik und des Vokabulars anderer Sprachen, d.h. sie enthält Informationen über die in einer Sprache transportierten Informationen. Mit Meta-Sprachen können also Sprachen, wie zum Beispiel Markup-Sprachen, definiert werden. Bei XML und dessen Vorgänger SGML (Standard Generalized Markup Language) handelt es sich um solche Meta-Sprachen zur Definition von Markup-Sprachen. Markup-Sprachen, auch als Auszeichnungssprachen bezeichnet, sind Sprachen, welche mittels eingestreuter Direktiven (Tags) die Struktur und/oder die Präsentation eines Dokumentes bestimmen. Markup-Sprachen erzeugen Dokumente immer für eine bestimmte Anwendung und werden meist für eine bestimmte Anwendung (z.B. Textformatierung) entworfen. Bei HTML handelt es sich um eine solche Auszeichnungssprache, wobei HTML selbst durch die Meta-Sprache SGML definiert wurde. Ein Dokument ist wiederum mit Hilfe einer Markup-Sprache definiert und beinhaltet letztendlich die Daten, ist also der Informationsträger. Ein Dokument besteht aus:

- Struktur (Markup-Elemente mit Attributen)
- Inhalt (die eigentlichen Informationen, d.h. Text- und Daten-Elemente)
- Präsentation (Definition der Ausgabe des Inhalts)

Als Beispiele sind RTF-Dokumente, Microsoft Word Dokumente, PDF-Dokumente und natürlich mit HTML definierte WWW-Seiten zu nennen. Der Nachteil von HTML liegt darin, dass der Inhalt optisch aber nicht semantisch strukturiert wird, d.h. der Inhalt verschmilzt mit der Präsentation. HTML Tags beschreiben nur, wie der Inhalt dargestellt werden soll, aber nicht, was er darstellt. So ist es nicht möglich die dargestellten Daten aus einem HTML-Dokument zu extrahieren, um sie an anderer Stelle wieder verwenden zu können. Es kommt also zu einem Informationsverlust im Sinne der Weiterverarbeitung der Daten.

Die folgende Abbildung verdeutlicht den Zusammenhang zwischen Metasprache, Markup-Sprache und Dokument.

Abbildung 2-8 Beziehung Meta-Sprache, Markup-Sprache, Dokument

2.1.2.3 XML Technologie

In diesem Kapitel sollen Syntax, Semantik und grundlegende Konzepte der Meta-Sprache XML anhand eines einfachen Beispiels dargestellt werden.

2.1.2.3.1 Wohlgeformte und gültige XML-Dokumente

Alle XML-Dokumente als konkrete Instanzen einer mit XML definierten Markup-Sprache haben eine einfache, universelle Grundstruktur, definiert durch die XML-Syntax. Nach dieser besteht ein XML-Dokument aus:

- einem Prolog mit einer ´Startanweisung´, die das Element als XML-Dokument identifiziert und den verwendeten Zeichensatz festlegt sowie einem optionalen Verweis auf eine DTD (siehe *Kapitel 2.1.2.3.2 - DTD (Document Type Definition)*)
- dem Hauptteil, bestehend aus einem Wurzelelement mit untergeordneten Elementen in Form einer Baumstruktur, wobei diese Elemente Attribute besitzen können
- einem optionalem Epilog, der Kommentare oder Verarbeitungsanweisungen enthalten kann

Ein Element enthält eine Anfangsmarke (´Start-Tag´), eine Endmarke (´End-Tag´) und einen zwischen diesen Marken geklammerten Inhalt. Der Inhalt kann aus weiteren strukturierten Elementen oder reinem Text bestehen. Elemente können auch Attribute besitzen, welche in der Anfangsmarke definiert werden. Ein Beispiel für ein solches Dokument liefert *Abbildung 2-9 Einfaches XML-Dokument*. In diesem Beispiel wird ein XML-Dokument dargestellt, wie es (im einfachsten Fall) bei einer elektronischen Bestellung auftreten könnte. Eine Ähnlichkeit mit HTML ist vorhanden, allerdings gelten folgende Regeln, die in HTML nicht zwingend sind:

- Elemente dürfen sich nicht überlappen
- leere Elemente können in Kurzform benutzt werden
- jedes Element muss mit einer Endmarke abgeschlossen werden
- Groß- und Kleinschreibung der Element- und Attributnamen ist entscheidend
- Attributwerte müssen in Anführungszeichen gesetzt werden

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Bestellung SYSTEM "Bestellung.dtd" >

<Bestellung>
        <Datum>14.04.2001</Datum>

        <Kunde>
                <Name>Roland Zack</Name>
                <EMail>Roland.Zack@t-online.de</EMail>
        </Kunde>

        <Artikel ArtikelNr="S0019">
                <Beschreibung>AMD Athlon 1100/256/30,5 – System</Beschreibung>
                <Menge>7</Menge>
                <Preis>2100 DM</Preis>
        </Artikel>

        <Artikel ArtikelNr="P00123">
                <Beschreibung>Agfa-Snapscan 310 Scanner</Beschreibung>
                <Menge>17</Menge>
                <Preis> 200 DM</Preis>
        </Artikel>

</Bestellung>
```

Abbildung 2-9 Einfaches XML-Dokument

Erfüllt ein Dokument diese Syntaxregeln der XML, wird es, wie das angeführte Beispiel, als ´wohlgeformt´ (´wellformed´) bezeichnet. Die Prüfung auf Wohlgeformtheit übernimmt ein ´nicht validierender Parser´. Näheres dazu später in *Kapitel 2.1.2.3.3 - Programmierschnittstellen und Parser*.

Die ´Wohlgeformtheit´ gibt an, ob ein Dokument den Syntaxregeln entspricht, nicht aber, ob die logische Struktur eines Dokumentes stimmt, d.h. ob es die richtigen Elemente in der richtigen Reihenfolge enthält, ob die Anzahl der Elemente stimmt, ob keine Elemente fehlen, usw. Hierzu wird die Semantik, d.h. Grammatik und das Vokabular der Sprache, in Form einer DTD (Document Type Definition) festgelegt. Stimmt ein Dokument mit dieser vorgegebenen Grammatik überein, wird es als ´gültig´ (´valid´) bezeichnet. Die Prüfung auf Gültigkeit übernimmt ein ´validierender Parser´. Ein gültiges Dokument ist gleichzeitig immer auch wohlgeformt.

2.1.2.3.2 DTD (Document Type Definition)

Die DTD beschreibt mit Hilfe der erweiterten Backus-Naur Form (EBNF) die Grammatik und das Vokabular einer XML-Sprache, d.h. sie ist ein Konstrukt, in dem sämtliche Element-Deklarationen, Attribut-Deklarationen, Notations-Deklarationen, Kardinalitäten, usw. zusammengefasst werden. Sie kann somit als Schablone auf ein XML-Dokument angewendet werden oder, um das Ganze objektorientiert auszudrücken, die DTD ist die Klasse und das XML-Dokument eine konkrete Instanz dieser Klasse. Eine solche DTD kann zum Beispiel zur Festlegung einer Corporate Identity für alle geschäftlichen Dokumente dienen. Mit der passenden Software (siehe *Kapitel 2.1.2.3.3 - Programmierschnittstellen und Parser*) können diese Dokumente ohne Probleme verarbeitet, gespeichert, publiziert und weltweit ausgetauscht werden. Eine DTD, welche das angeführte Beispiel in ein gültiges Dokument ´verwandeln´ würde und die entsprechende Markup Sprache definiert, könnte wie in *Abbildung 2-10 Einfache DTD* aussehen. In dieser DTD werden für konkrete Instanzen folgende Vorgaben gemacht:

- Das Element ´Bestellung´ besteht aus folgenden Elementen, die in dieser Reihenfolge aufgeführt werden müssen: ´Datum´, ´Kunde´ und eine nichtleere Menge von Artikeln.
- Das Element ´Datum´ besteht aus einfachem Text.
- Das Element ´Kunde´ besteht aus Name und optionaler E-Mail-Adresse, welche Textelemente sind.
- Das Element ´Artikel´ besteht aus einer Beschreibung, einer Menge und einem Preis, welche jeweils Textelemente sind.
- Jeder Artikel hat zwingend ein Attribut ´Artikelnummer´.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT Bestellung      (Datum, Kunde, Artikel+)     >
<!ELEMENT Datum           (#PCDATA)                    >
<!ELEMENT Kunde           (Name, EMail?)               >
<!ELEMENT Name            (#PCDATA)                    >
<!ELEMENT EMail           (#PCDATA)                    >
<!ELEMENT Artikel         (Beschreibung, Menge, Preis) >
<!ATTLIST Artikel         ArtikelNr ID #REQUIRED       >
<!ELEMENT Beschreibung    (#PCDATA)                    >
<!ELEMENT Menge           (#PCDATA)                    >
<!ELEMENT Preis           (#PCDATA)                    >
```

Abbildung 2-10 Einfache DTD

2.1.2.3.3 Programmierschnittstellen und Parser

XML Parser sind Bestandteile von Anwendungsprogrammen, welche XML-Dokumente verarbeiten. Das heißt, XML Parser analysieren Dokumente und stellen der Anwendung die benötigten Informationen über diese zur Verfügung. Mittels einheitlicher, plattform- und sprachunabhängiger APIs kann der Parser auf die Dokumente und die darin enthaltenen Daten zugreifen. Hierbei existieren zwei grundlegende Schnittstellen: Die objektmodell-orientierte DOM-Schnittstelle und die ereignisorientierte SAX-Schnittstelle. Zur Verdeutlichung dient das folgende Schaubild:

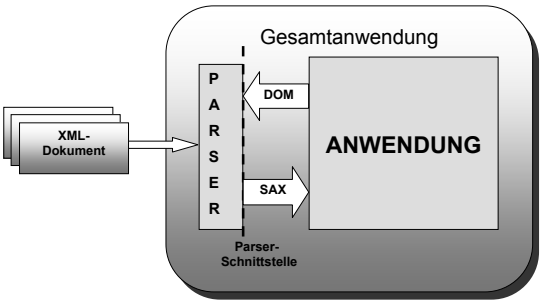

Abbildung 2-11 Programmierschnittstellen und Parser

Die DOM-Schnittstelle, auch als ´baumorientierte Schnittstelle´ bezeichnet, erlaubt, wie auch SAX, den standardisierten Zugriff auf beliebige HTML- und XML-Dokumente. Sie bietet Methoden zum Lesen und zur Manipulation der Baumstruktur eines Dokumentes. Ein DOM-Parser verarbeitet (parst) zunächst das gesamte Dokument und erstellt intern einen ´Elementebaum´, der auch als ´Syntaxbaum´ oder ´DOM-Baum´ bezeichnet wird und als Repräsentant des Dokumentes fungiert. Die an den Parser gekoppelte Anwendung kann nun über einheitliche Schnittstellen diesen Baum durchwandern und manipulieren, indem sie ihren Zugriff auf einzelne Knoten, sowie deren Vorgänger und Nachfolger, geschickt nutzt. Das allgemeine Vorgehen bei der Nutzung der DOM-Schnittstelle ist also die Objektmanipulation auf folgende Art und Weise:

1. Parser instanziieren
2. XML-Dokument vollständig lesen und dabei DOM-Baum aufbauen
3. Resultat-Baum durchwandern, durchsuchen und die Informationen manipulieren

Der Vorteil gegenüber der ereignisorientierten SAX-Schnittstelle liegt in der Möglichkeit der Navigation und Manipulation des Baumes, d.h. es ist hier möglich die Baumstruktur vorwärts und rückwärts zu durchwandern. Der große Nachteil dieser Methode besteht darin, dass das gesamte Dokument geparst werden muss, bevor auf dessen Inhalt zugegriffen werden kann und dieser Vorgang bei großen Dokumenten und somit großen Resultat-Bäumen sehr speicherintensiv ist[1]. Um dieses Problem zu lösen, wurde in der XML-Gemeinde die SAX-Schnittstelle vorgeschlagen und unter dem Motto “Don´t call the DOM, the parser calls you!“ standardisiert.

Der DOM-Baum des Beispieles im *Kapitel 2.1.2.3.1 – Wohlgeformte und gültige XML-Dokumente* stellt sich folgendermaßen dar:

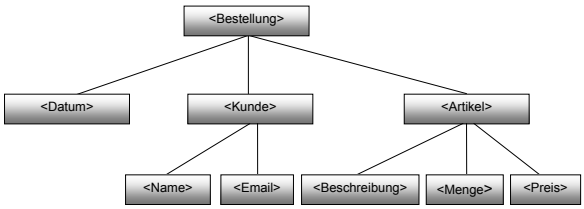

Abbildung 2-12 DOM-Baum

Die SAX-Schnittstelle, auch als ´ereignisorientierte Schnittstelle´ bezeichnet, erlaubt, wie auch DOM, den standardisierten Zugriff auf beliebige XML-Dokumente, wobei jedoch nicht der DOM-Baum aufgebaut und anschließend bearbeitet wird. Ein SAX-Parser liest das XML-Dokument sequentiell und ruft bei Ereignissen, wie zum Beispiel ´Dokumentenanfang´ oder ´End-Tag´ entsprechende Methoden (Event-Handler-Methoden) auf, deren Rümpfe der Anwendungsprogrammierer seinen Bedürfnissen entsprechend gestalten kann. Das allgemeine Vorgehen bei der

[1] Dieses Problem tritt insbesondere beim Einsatz von XML in Zusammenhang mit Streaming-Technologien auf.

Nutzung der SAX-Schnittstelle ist also die Ereignisbehandlung auf folgende Art und Weise:

1. Parser instanziieren
2. Event-Handler instanziieren
3. Event-Handler beim Parser registrieren
4. Dokument sequentiell einlesen
5. bei jedem Ereignis den entsprechenden Event-Handler informieren

Der Vorteil gegenüber der baumorientierten DOM-Schnittstelle liegt in der vielfach höheren Verarbeitungsgeschwindigkeit der SAX-Parser, da kein vollständiger DOM-Baum aufgebaut werden muss und die an den Parser gekoppelte Anwendung die für sie unwichtige Teile des Dokumentes ignorieren kann. Weiterhin besteht hierdurch die Möglichkeit sehr große Dokumente und Streams zu verarbeiten. Allerdings ist die SAX-Schnittstelle aufgrund der fehlenden Möglichkeit der Baum-Manipulation nicht für jeden Anwendungsfall geeignet.

2.1.2.3.4 XML – Dokumente oder Daten?

Eine konkrete XML-Instanz, welche meist als XML-Dokument bezeichnet wird, fällt immer in eine der beiden Kategorien:

- XML-Dokument (im engeren Sinne)
- XML-Daten

XML unterscheidet nicht, ob eine Instanz ein Dokument oder Daten enthält, bei der Benutzung bestimmter Werkzeuge zur Bearbeitung von XML-Instanzen ist diese Unterscheidung jedoch wichtig. Meist kann die Kategorie durch bloßes Betrachten bestimmt werden:

Typische XML-Dokumente werden primär vom Menschen geschrieben und gelesen. Hierzu zählen zum Beispiel Bücher, Zeitschriften, Webseiten usw. Solche Dokumente enthalten hauptsächlich Fließtext, einige Überschriften, Tabellen oder Bilder. Das heißt der Inhalt dieser Elemente wird als reiner Text interpretiert.

Typische XML-Daten sind Rechnungen, Bestellungen, Kataloge, Preislisten und andere strukturierte Informationen, die primär für die maschinelle Verarbeitung und den Datenaustausch gedacht sind. Solche Dokumente enthalten oft ´hochstrukturierte´ Daten, die bestimmte Integritätsbedingungen erfüllen und typisiert sein müssen. (siehe dazu *Kapitel 2.1.2.3.6 – XML Schema*).

2.1.2.3.5 XML Namensräume

Innerhalb eines XML-Dokumentes muss jeder Element- und Attributname eine eindeutige Bedeutung besitzen, ansonsten ist das Dokument nicht gültig. Da man aber sprechende Namen für Elemente und Attribute verwenden sollte, kann dies bei komplexeren Systemen oft zu Mehrdeutigkeiten und somit zu Namenskollisionen führen. So kann zum Beispiel ein Tag *<Titel>* sowohl für die Anrede als auch für den Titel einer Veröffentlichung stehen. Zur Auflösung dieser Mehrdeutigkeiten von Metainformationen, die durch die rasant steigende Anzahl von DTDs und XML Schemata unausweichlich wird, dienen XML Namespaces.

Dazu qualifiziert man die mehrdeutigen Namen durch Bindung an URIs. Ein Namensraum ist eine eindeutig identifizierte Sammlung von Namen für Elemente und Attribute, qualifiziert durch einen URI. Ein Beispiel soll dies verdeutlichen:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<bestellung      xmlns:sender="http://www.sender.com"

                 xmlns:empfänger="http://www.empfänger.de">

        <sender:artikel artikelnr="S0019">
                <beschreibung>AMD Athlon 1100/256/30,5 – System</beschreibung>
                <menge>7</menge>
                <preis>2100 DM</preis>
        </sender:artikel>

        <empfänger:artikel>
                <beschreibung> AMD Athlon 1100/256/30,5 – System</beschreibung>
                <menge>7</menge>
                <währung>DM</währung>
                <preis>2100</preis>
        </empfänger:artikel>

</bestellung>
```

Abbildung 2-13 Einfaches Namensraum-Beispiel

Hierbei werden innerhalb des Elementes *<bestellung>* zwei Namensräume in der Form *<elementname* xmlns:*namensraumprefix* = "*uri*"> deklariert. Die Verwendung eines Elementes geschieht durch Voranstellen des entsprechenden Präfix und die Trennung durch einen Doppelpunkt. So ist es, wie in diesem Beispiel beim Element *<artikel>*, möglich, zwei Elemente bzw. Attribute gleichen Namens in unterschiedlichen Kontexten zu benutzen.

2.1.2.3.6 XML Schema

Das Konzept der DTD stammt aus der dokumentenorientierten Welt der SGML. XML wird aber nicht nur zur Dokumentenverwaltung, sondern auch für den Datenaustausch benutzt (siehe *Kapitel 2.1.2.3.4 – XML - Dokumente oder Daten?*). In DTDs können dagegen nur Textelemente definiert werden, was für den Datenaustausch nicht ausreichend ist. XML Schema erweitert die Funktionalität der DTDs abwärtskompatibel und unterstützt viele gängige Datentypen wie String, Real, Boolean, Date, Language, sowie selbstdefinierte Typen, Vererbungsmechanismen, Integritätsbedingungen, Listen, Referenzen usw. Ein weiterer Vorteil der XML Schema Sprache besteht darin, dass sie selbst in XML formuliert ist und somit im Gegensatz zur DTD keinen speziellen Parser benötigt. Die Nachteile von XML Schema gegenüber der DTD sind die mittlerweile sehr große Verbreitung und Kenntnis von DTDs, die Vielzahl von Werkzeugen und bereits auf Basis von DTDs definierten Sprachen. Bei Benutzung von XML Schema wird ein Dokument nicht mehr auf Gültigkeit (siehe *Kapitel 2.1.2.3.1 – Wohlgeformte und gültige XML-Dokumente*) sondern auf ´Schema-Konformität´ geprüft. Weiterhin existieren im Gegensatz zu DTDs drei Ebenen der Definition:

- Typ-Definition (erfolgt im Schema)
- Element- und Attribut-Deklarationen, die diese Typen verwenden (erfolgt ebenfalls im Schema)
- Element-Definitionen, d.h. Element-Instanzen (erfolgt im Dokument)

Das folgende kurze Beispiel soll die Funktionsweise von XML Schema verdeutlichen. Zuerst wird im Schema ein neuer Typ namens *nameT* definiert, der sich aus zwei Strings und einem optionalen Datum zusammensetzt. Im zweiten Schritt werden zwei Elemente auf Basis dieses Types deklariert, die im dritten Schritt im Dokument instanziiert werden.

```
<xsd:schema xmlns=http://www.w3.org/2000/08/XMLSchema>

        <!-- Den Typ ´NameT´ definieren -->
        <xsd:complexType name="nameT">
                <xsd:sequence>
                        <xsd:element name="vorname"   type="xds:string"/>
                        <xsd:element name="nachname" type="xds:string"/>
                </xds:sequence>
                <xsd:attribute name="geburtsdatum" type="xsd:date" use="optional"/>
        </xsd:complexTyp>
```



```
        <!-- Den neuen Typ in zwei Elementdeklarationen verwenden -->
        <xsd:element name="name" type="nameT"/>
        <xsd:element name="autor " type="nameT"/>

</xsd:schema>
```

Abbildung 2-14 Einfaches XML Schema

```
<!-- Elemente vom Typ nameT -->
<name geburtsdatum="1955-02-02">
   <vorname> Roland </vorname>
   <nachname> Zack </nachname>
</name>

<autor>
   <vorname> Eva </vorname>
   <nachname> Eisen </nachname>
</autor>
```

Abbildung 2-15 Benutzung der im Schema deklarierten Elemente

2.1.2.3.7 XSL – Transformation und Darstellung

In *Kapitel 2.1.2.1 – Warum XML?* wurde der Vorteil der strikten Trennung zwischen dem Inhalt und dessen Präsentation hervorgehoben und gezeigt, dass ein XML-Dokument keinerlei Informationen über die Präsentation der transportierten Daten beinhaltet. Hierzu dient die selbst in XML definierte Sprache XSL (Extensible Stylesheet Language), welche als entkoppelte Präsentationsebene fungiert.

XSL ermöglicht es, ein XML-Dokument auf verschiedenen Medien darzustellen, oder um es anders auszudrücken, durch XSL werden verschiedene Sichten auf ein XML-Dokument ermöglicht (´Views-Konzept´). So kann zum Beispiel ein XML-Dokument, welches aktuelle Börsennachrichten enthält, im Internet als HTML-Dokument publiziert werden, als PDF zur Druckausgabe zur Verfügung gestellt werden, an ein WAP-Handy in Form eines WML-Dokumentes geschickt werden, in Sprachausgabe umgewandelt werden oder im reinen Textformat als E-Mail oder SMS versendet werden. Weitere Anwendungen sind das automatische Generieren einer Antwort aus einer Anfrage im Online-Shopping Bereich, die Transformation von XML-Dokumenten in EDI-Nachrichten[1], die Darstellung eines Dokumentes in

[1] EDI steht für ´Electronic Data Interchange´ und es handelt sich hierbei um ein weit verbreitetes Datenaustauschformat für Geschäftsnachrichten.

unterschiedlichen Anwendungskontexten (extern, intern, Management, Mitarbeiter), die Vereinheitlichung von Dokumenten, das Filtern von Informationen usw. XSL besteht aus zwei Sprachen: aus der Transformationssprache XSLT (Extensible Stylesheet Language Transformations) und der Formatierungssprache XSL FO (Extensible Stylesheet Language Formatting Objects).

XSLT ist eine deklarative Sprache, mit der beschrieben wird, wie ein XML-konformes Dokument in ein anderes, meist auch XML-konformes Dokument, umgewandelt werden soll. Die Umsetzung kann das alte Vokabular übernehmen, aber auch verändern oder vollständig ersetzen. Die bekanntesten Anwendungen sind die Überführung von semantischen XML-Dokumenten in Anzeigestruktur-Dokumente wie HTML, um diese grafisch aufbereitet in einem Browser zu präsentieren, sowie das Konvertieren von XML-Dokumenten zwischen verschiedenen Vokabularen, also zum Beispiel zwischen dem internen Format der Firma A und dem der Firma B. Eine beispielhafte Anwendung wäre des automatische Generieren eines ´Inhaltsverzeichnis-XML-Dokumentes´ aus den Kapitelüberschriften eines sehr großem XML-Dokumentes.

Die Transformation eines XML-konformen Dokumentes in ein anderes XML-konformes Dokument geschieht nach folgendem Prinzip: Ein XSLT-Prozessor liest das Quell-Dokument sequentiell ein und vergleicht in jedem Knoten, ob dieser einem in der Stylesheet-Datei angegebenen, Muster entspricht. Trifft dies zu, wird die entsprechende, auch im Stylesheet angegebene Aktion ausgeführt. Ansonsten wird das Element kopiert.

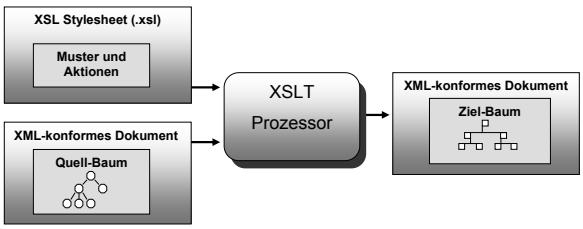

Abbildung 2-16 Prinzip der XSLT

Eine XSL Stylesheet-Datei besteht aus einem Satz von Regeln, welche festlegen, wie das Quelldokument, d.h. der Quellbaum, in das Zieldokument, also den Zielbaum, konvertiert werden soll. Eine solche Konstruktionsregel gliedert sich in folgende Teile:

- Muster (pattern)
  Das Muster gibt an, auf welche Bezeichner innerhalb des XML-Dokumentes sich die darauf folgende Aktion beziehen soll, das heißt, findet der Parser das angegebene Muster, wird die entsprechende Aktion ausgeführt.
- Aktion (action)
  Wurde das angegebene Muster erkannt, wird die darauf folgende Aktion ausgeführt. Diese Aktion kann aus der entsprechenden Umwandlung des betroffenen Elementes in die deklarierte Ausgabeform, d.h. das korrespondierende Element des Zielbaumes (Template, Schablone), oder aus einer dynamischen Anweisungen, zum Beispiel der Aktivierung eines Skriptes, bestehen.

XSL FO ist eine Sprache, mit der beschrieben wird, wie ein Formatierer Inhalte auf dem Ausgabemedium platzieren soll. Dies bedeutet, hier wird ein XML-konformes Dokument in ein nicht XML-konformes Dokument, wie zum Beispiel ein Format für Printmedien wie PDF, umgewandelt. Es können hierbei verschiedene Einstellungen zu den Rändern, Füll-, Hintergrund- und Schrifteigenschaften usw. eingestellt werden. XSL FO zielt nicht auf die Fähigkeiten eines Programmierers, sondern die eines Designers oder Layouters ab[1].

Die Transformation eines XML-konformen Dokumentes in ein nicht XML-konformes Dokument für Printmedien geschieht nach folgendem Prinzip: Das Quelldokument wird hierzu zuerst per XSLT in einen FO-Baum umgewandelt und dieser dann mit einem FO-Prozessor in das nicht XML-konforme Formate übersetzt (PDF, RTF, Excel, Powerpoint usw.).

[1] Mittlerweile existieren eine Reihe von Werkzeugen, welche es erlauben das gewünschte Design zu gestalten und anschließend die XSL und XSL FO Dokumente automatisch zu erzeugen.

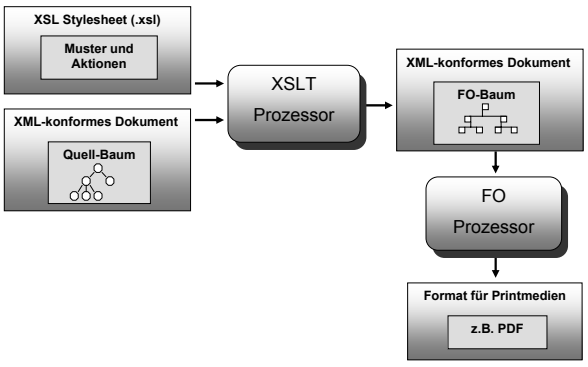

Abbildung 2-17 Prinzip der XSL FO

2.1.3 SOAP (Simple Object Access Protocol)

Wie bereits in *Kapitel 2.1.1.6 - Web Services Standards* angeführt, dient das Simple Object Access Protocol (SOAP) als XML-basiertes Messaging-Protokoll. Es handelt sich um ein leichtgewichtiges, interoperables Protokoll zum Informationsaustausch in einer dezentralisierten, verteilten Umgebung, welches sowohl RPCs als auch asynchrones Messaging ermöglicht.

SOAP ist im Gegensatz zu IIOP[1], DCOM[2] oder RMI[3] kein verteiltes Komponentenmodell. Es muss nicht zwingend in Verbindung mit verteilten Objekten und Methodenaufrufen benutzt werden. Weiterhin besitzt es keinerlei Ansätze in den Bereichen Object Discovery, Distributed Garbage Collection, Authentifizierung und Verschlüsselung. Oberste Ziele dieses Protokolls sind Interoperabilität, Einfachheit und Erweiterbarkeit. Die genannten Punkte können durch entsprechende Erweiterungen auf SOAP aufgesetzt werden: UDDI dient dem Object Discovery, zur Verschlüsselung kann HTTPS verwendet werden und Authentifizierungs-informationen können über den Header mitgeschickt werden. SOAP ist vom zugrunde liegenden Kommunikationsprotokoll unabhängig, weshalb der Austausch der XML Daten über beliebige Transportprotokolle erfolgen kann. Für RPCs sind im Internet-Bereich meist HTTP/HTTPS und für die asynchrone Kommunikation SMTP vorgesehen. Die SOAP-Spezifikation besteht aus folgenden Teilen:

- SOAP Envelope
  Dieser Teil definiert ein Framework für die Beschreibung des Inhaltes einer SOAP-Nachricht, wer diese verarbeiten soll, wie dies geschehen soll und ob die Verarbeitung obligatorisch oder optional ist.
- SOAP Encoding Rules
  Die SOAP-Kodierungsregeln dienen der Beschreibung anwendungsspezifischer Datentypen und ermöglichen damit, Objektinstanzen zu serialisieren und auszutauschen.

[1] IIOP (Internet Inter ORB Protocol) ist das CORBA-Protokoll für die Kommunikation im Internet.

[2] Die Distributed Components (DCOM) Architektur ist das verteilte Komponentenmodell der Microsoft-Plattform.

[3] RMI (Remote Method Invocation) ist das verteilte Komponentenmodell der Java-Plattform.

- SOAP RPC Representation
  Dieser Teil der Spezifikation stellt Vorschriften für die Repräsentation von entfernten Prozeduraufrufen und deren Antworten auf.
- SOAP Protocol Bindings
  An dieser Stelle wird definiert, wie SOAP-Nachrichten in HTTP und anderen Transportprotokollen transportiert werden.

Eine SOAP-Nachricht ist ein XML-Dokument, welches sich wie folgt zusammensetzt:

- SOAP-Envelope
  Der SOAP-Envelope umfasst die gesamte SOAP-Nachricht und besteht aus einem optionalen SOAP-Header und einem obligatorischem SOAP-Body.
- SOAP-Header
  Der Header enthält zusätzliche Informationen für den Empfänger der Nachricht, welche für die Verarbeitung der Nachricht notwendig oder hilfreich sind. Dieser flexible Mechanismus erlaubt die einfache Erweiterbarkeit der Kommunikation, da solche Erweiterungen ohne Kenntnisnahme durch den Kommunikationspartner geschehen können. Typische Anwendungsbeispiele sind hier Transaktionsmanagement, Routing-Mechanismen und Authentifizierung.
- SOAP-Body
  Im Body der Nachricht sind die eigentlichen Nutzdaten enthalten, d.h. zum Beispiel die Parameter eines entfernten Aufrufs.

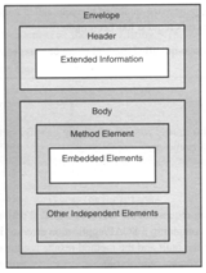

Abbildung 2-18 Aufbau einer SOAP Nachricht (Quelle: [SKSM00], S.81)

Als Beispiel für die SOAP-Kommunikation sollen die folgenden Abbildungen dienen, welche eine mögliche Verwendung einer Serveranwendung zeigen, die aktuelle Wertpapierkurse bereitstellt. Der SOAP-Request in der ersten Abbildung dient dem entfernten Aufruf der Methode *´getLastTradePrice´*, welche als Parameter die Kennung des Wertpapiers erwartet (in diesem Fall IBM) und dessen aktuellen Kurs zurückliefert. Eine mögliche Antwort zeigt die zweite Abbildung: die Serveranwendung liefert ´139.7´ als aktuellen Kurswert zurück.

```
POST /soap/servlet/rpcrouter HTTP/1.1
Host: stockprovider:8080
Content-Type: text/xml; charset="utf-8"
Content-Length: 318
SOAPAction: ""

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope
 SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding>

        <SOAP-ENV:Body>
                <m:getLastTradePrice xmlns:m="urn:xmltoday-delayed-quotes">
                        <symbol>IBM</symbol>
                <m:getLastTradePrice>
        </SOAP-ENV:Body>

<SOAP-ENV:Envelope>
```

Abbildung 2-19 Beispiel für einen SOAP-Request

```
http/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 307

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope
 SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding>

        <SOAP-ENV:Body>
                <m:getLastTradePriceResponse xmlns:m="urn:xmltoday-delayed-quotes">
                        <price>139.7</price>
                <m:getLastTradePrice>
        </SOAP-ENV:Body>

<SOAP-ENV:Envelope>
```

Abbildung 2-20 Beispiel für eine SOAP-Response

Die Kommunikation via SOAP soll anhand des folgenden Szenarios verdeutlicht werden:

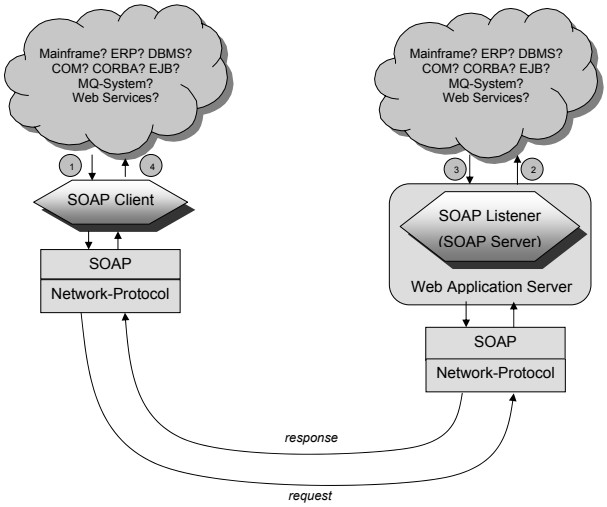

Abbildung 2-21 SOAP Kommunikation

Die Grundanforderungen an einen Netzwerkknoten, welcher als Service Provider bzw. Service Requestor agieren will, sind die Fähigkeit, SOAP-Nachrichten zu erzeugen und zu verarbeiten. Der Service Requestor erreicht dies durch Zugriff auf entsprechende APIs, der Service Provider wird in den meisten Fällen durch einen SOAP-Listener realisiert, der in einem Web Application Server auf SOAP Requests horcht. Die SOAP-Kommunikation verläuft nach folgendem Schema:

- Im Punkt (1) erzeugt die Service Requestor Applikation eine SOAP-Nachricht. Diese Nachricht enthält den Request, welcher die entsprechende Operation des Service Providers aufruft. Der Request kann sowohl ein RPC als auch ein einfaches XML-Dokument sein. Der Service Requestor übergibt die Nachricht

  zusammen mit der Netzwerkadresse des Service Providers an die SOAP Infrastruktur, d.h. zum Beispiel eine SOAP-Client Laufzeitumgebung. Diese Laufzeitumgebung interagiert mit der darunter liegenden Netzwerkinfrastruktur (zum Beispiel HTTP), um die SOAP-Nachricht über das Netzwerk zu versenden.
- Im Punkt (2) übergibt die Netzwerkinfrastruktur die Nachricht an die SOAP Laufzeitumgebung des Service Providers, die die Nachricht an die dahinter liegende Applikation weiterleitet. Die Laufzeitumgebung ist für die Konvertierung der SOAP Nachricht in programmiersprachenabhängige Objekte verantwortlich. Diese Konvertierung wird durch Encoding Schemas, welche in der Nachricht enthalten sind, und Databinding-Frameworks (siehe *Kapitel 3.4.3.3 – Castor 0.9.3*) unterstützt.
- Nachdem der Request verarbeitet wurde, wird im Punkt (3) eine SOAP Nachricht erzeugt, die die Antwort auf diesen Request enthält und auf dem gleichen Weg an den Service Requestor übergeben wird. Im Falle HTTP wird der Nachricht/Antwort Mechanismus des Protokolls ausgenutzt.
- Im Punkt (4) erhält der Service Requestor die Antwort auf seine Anfrage. Die Nachricht wird auf gleichem Weg durch die SOAP Laufzeitumgebung geführt, welche diese in Objekte der Zielprogrammiersprache konvertiert und sie dem Service Requestor übergibt.

Dieses Beispiel betrachtet nur das am weitesten verbreitete Anfrage/Antwort Szenario. SOAP kann jedoch mit jeder beliebigen Übertragungsart verwendet werden:

- request/response - synchron
- request/response - asynchron
- one-way messaging - no response
- notification - push-style response
- publish/subscribe

Zusammenfassend sollen an dieser Stelle noch einmal die Vor- und Nachteile gegenüber dem Querschnitt[1] der traditionellen RPC- und Messaging-Protokolle aufgezeigt werden. Die Vorteile:

[1] Auch CORBA ist programmiersprachenunabhängig.

- keine Probleme mit Firewalls, da Kommunikation über HTTP
- Client und Server in beliebigen Programmiersprachen implementierbar
- Plattformunabhängigkeit
- keine Lizenzkosten; nicht proprietär
- unterschiedliche Serialisierungsversionen voll kompatibel
- Streaming über SAX möglich
- vom Menschen lesbar (Fehleranalyse, Systemtests)

Die Nachteile dieses Protokolls, welches, zumindest im jetzigen Stadium, sicher nicht für jedes beliebige Einsatzszenario geeignet ist, sind:

- Geschwindigkeitsprobleme (bis Faktor 1:10), aufgrund der Übertragung im Textformat, Parsen und Validierung
- keine Integration in das OO-Modell der Applikation (Mapping notwendig)
- keine Transaktionslogik, da HTTP basiert

2.1.4 WSDL (Web Services Description Language)

Damit eine Applikation einen Web Service nutzen kann, muss dessen Schnittstelle präzise beschrieben sein. Dazu dient die XML-basierte Sprache WSDL (Web Services Description Language), welche eine analoge Rolle zu der in der CORBA-Welt bekannten Interface Definition Language (IDL) spielt. Die Beschreibung muss Details über das verwendete Protokoll, Host- und Portnummern, die aufrufbaren Operationen, deren Ein- und Ausgangsparameter, möglicherweise auftretende Ausnahmen usw. enthalten. Ein WSDL Dokument basiert auf den folgenden Entitäten:

- Service
  Ein Service verbindet verschiedene Ports, welche semantisch oder fachlich zusammengehören. Die zusammengefassten Ports kommunizieren nicht mit- bzw. untereinander und sollen dem WSDL-Nutzer die Auswahl unter den Ports ermöglichen.
- Port
  Ein Port legt die konkrete Netzwerkadresse des Services an, unter der ein WSDL-Nutzer diesen erreichen kann. Daneben wird auch eine Referenz auf eine konkrete Bindungsdefinition gelegt.

- Binding
  Mittels Binding wird die Netzwerkadresse mit konkreten Protokollen und Datenformaten assoziiert bzw. diese Informationen mit den abstrakten Operationsdefinitionen in Einklang gebracht.
- PortType
  Ein PortType stellt eine abstrakte Sammlung von Operationen mit unterschiedlichen Kommunikations-Semantiken dar.
- Operation
  Diese Entität stellt eine abstrakte Beschreibung einer vom Web Service zur Verfügung gestellten Aktion dar. Sie führt alle notwendigen Definitionen bzgl. der Datenpakete zu einer Operation zusammen. Es können Input, Output und Fehlermeldungen zu einer Operation assoziiert werden.
- Message
  Diese Entität stellt eine abstrakte typisierte Definition der auszutauschenden Daten dar.

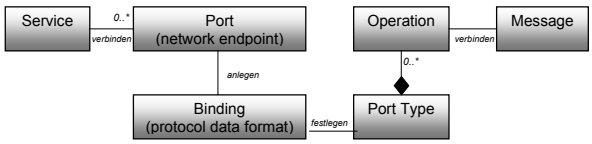

Abbildung 2-22 WSDL Entitäten und deren Beziehungen

Nachfolgend wird das zum SOAP Beispiel Web Service in *Kapitel 2.1.3 - SOAP (Simple Object Access Protocol)* passende WSDL Dokument aufgeführt, um einen Eindruck dieser Beschreibungssprache zu erhalten.

```
<?xml version="1.0"?>
<definitions name="StockQuote"
  targetNamespace="http://www.myserver.com/services/stockquote.wsdl"
  xmlns:tns="http://www.myserver.com/stockquote.wsdl"
  xmlns:xsd1=http://www.myserver.com/stockquote.xsd
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns=http://schemas.xmlsoap.org/wsdl/>

    <types>
        <schema targetNamespace=xmlns=http://www.w3.org/2000/10/XMLSchema>
            <element name="TradePriceRequest">
                <complexType>
                    <all><element name="tickerSymbol" type="string"/></all>
                </complexType>
            </element>
            <element name="TradePrice">
                <complexType>
                    <all><element name="price" type="float"/></all>
                </complexType>
            </element>
        </schema>
    </types>

    <message name="getLastTradePriceInput">
        <part name="body" element=xsd1:TradePriceRequest/>
    </message>

    <message name="getLastTradePriceOutput">
        <part name="body" element=xsd1:TradePrice/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="getLastTradePrice">
            <input message="tns:getLastTradePriceInput"/>
            <output message="tns:getLastTradePriceOutput"/>
        </operation>
    </portType>


<binding name="StockQuoteBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport=http://schemas.xmlsoap.org/soap/http/>
        <operation name="getLastTradePrice">
            <soap:operation soapAction=http://www.myserver.com/getLastTradePrice/>
            <input>
                <soap:body use="literal"
                    namespace=http://www.myserver.com/stockquote.xsd
                    encodingStyle=http://schemas.xmlsoap.org/soap/encoding/>
            </input>
            <output>
                <soap:body use="literal"
                    namespace=http://www.myserver.com/stockquote.xsd
                    encodingStyle=http://schemas.xmlsoap.org/soap/encoding/>
            </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <documentation>Service that delivers the latest stockquotes</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
            <soap:address location=http://www.myserver.com/stockquote/>
        </port>
    </service>
</definitions>
```

Abbildung 2-23 WSDL Dokument

Wie man sieht ist dieses (fast schon minimale) WSDL-Dokument sehr umfangreich und komplex. Es bietet sich daher an, sofern die Web Services Implementierung bereits besteht, entsprechende Werkzeuge zu benutzen, um es aus der

Implementierung zu erzeugen. Ein weiterer wichtiger Weg ist der umgekehrte: Man kann mit speziellen Werkzeugen aus einem bereits vorhandenen WSDL-Dokument Code generieren. Hierbei gilt es zwei Möglichkeiten zu unterscheiden: Die erste Möglichkeit besteht darin, einen so genannten ´Client Service Proxy´ zu generieren, welcher nun die Kommunikation mit dem Web Service kapselt und dem Entwickler ermöglicht, mit dem Web Service wie mit einem lokalen Objekt zu arbeiten. Die zweite Möglichkeit besteht darin, ein ´Web Service Implementation Template´ zu erzeugen, zu dem der Anwendungsentwickler nur noch die applikationsspezifischen Details implementieren muss.

Die allgemeine Definition eines Web Service kann nun durch eine technische Definition ergänzt werden:

Ein Web Service ist eine Softwarekomponente mit den folgenden Eigenschaften:

- Der Zugriff geschieht über eine SOAP-basierte Schnittstelle.
- Die Schnittstelle wird über ein WSDL Dokument beschrieben.

2.1.5 UDDI (Universal Description, Discovery, Integration)

Web Services lassen sich mit SOAP aufrufen und mit WSDL beschreiben. Als drittes Grundelement bedarf es noch eines Instrumentariums zum Publizieren und Suchen dieser Dienste, d.h. es fehlt ein weltweit zugreifbarer Verzeichnisdienst. Was der Namensdienst für CORBA oder die Registry für RMI sind, ist das UDDI (Universal Description, Discovery, Integration) für Web Services. Es handelt sich hierbei primär um eine globale Business-Registry, die mehrere Unternehmen gemeinsam betreiben. UDDI erlaubt die Suche nach Unternehmen, Branchen und Web Services.

Wie bei allen Informationssystemen auf der Ebene von Objekten existieren auch bei UDDI drei Phasen:

- publish
  Der Service Provider stellt Informationen über sich und seine angebotenen Dienste in Form von WSDL-Dokumenten in die Datenbasis der UDDI Registry ein.
- find
  Der Service Requestor sucht innerhalb der UDDI Registry einen geeigneten Dienst und kann über verschiedene Einstiegspunkte gewünschte Informationen abfragen. Folgende Kategorien existieren:

- white pages – Informationen über Unternehmen
- yellow pages –Branchenverzeichnis, das eine branchen-spezifische Suche nach Unternehmen ermöglicht, also eine Katalogisierung nach Produkten und Diensten darstellt
- green pages – Informationen zu Geschäftsmodell, Geschäftsprozessen und Services eines Unternehmens

- bind
  Der Service Requestor hat einen geeigneten Dienst und seinen Service Provider gefunden und interagiert nun mit diesem.

UDDI kann neben der Integration von Geschäftspartnern auch innerhalb eines Unternehmens verwendet werden, um interne Geschäftsfunktionalitäten, welche als Web Services offen gelegt sind, zu organisieren. So können zum Beispiel Programme geschrieben werden, die einen Dienst für eine gegebene Geschäftsfunktionalität aufsuchen, wodurch gegenüber den bisherigen statischen Workflows eine enorme Flexibilität, insbesondere bei häufigen Aktualisierungen, erreicht wird.

Programmtechnisch erfolgt der Zugriff auf UDDI-Verzeichnisse über SOAP-Nachrichten. Die ´Inquiry API´ enthält zu diesem Zweck Methoden, um Informationen über Unternehmen und Web Services abzufragen. Auf der anderen Seite ermöglicht die ´Publishers API´ es, entsprechende UDDI Einträge abzulegen und zu modifizieren. Rein technisch gesehen besteht UDDI aus einer Datenbank, der so genannten UDDI-Registry. Die in dieser Datenbank abgelegten Informationen besitzen eine fest vorgegebene Struktur und jeder Nutzer ist für seine Daten und deren Aktualität selbst verantwortlich. Die Nutzung dieser Datenbank kann entweder über spezielle Suchmaschinen erfolgen oder als Web Service in die eigene Anwendung integriert werden. Derzeit existieren UDDI Registries auf den Web-Seiten von Ariba, IBM, Microsoft und HP.

Die zur Bereitstellung der Infrastruktur benötigten Informationen werden, wie in *Abbildung 2-24 UDDI Datenstruktur* dargestellt, in vier Bereiche unterteilt. Auf der obersten Ebene (Business Entity Ebene) werden allgemeine Informationen über das Unternehmen abgelegt. Dazu gehören u.a. der Name des Unternehmens, Kontakte und eine Beschreibung. Auf der nächsten Ebene (Business Service Ebene) erfolgt eine Unterteilung des Unternehmens in Geschäftsbereiche oder andere Kategorien,

wie zum Beispiel Produktsparten oder geografische Regionen. Ein Business Service kann aus mehreren Binding Templates bestehen, welche die Beschreibung der Kommunikationsschnittstelle, also des Web Services, beinhalten. Um detailliertere Informationen zu erhalten, können diese Binding Templates Referenzen auf so genannte ´tModels´ enthalten, die die unterste Ebene darstellen.

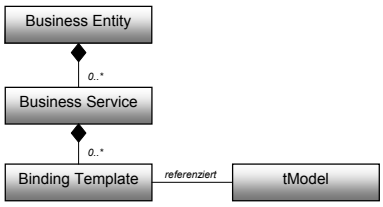

Abbildung 2-24 UDDI Datenstruktur

Abschließend kann nun das in *Kapitel 2.1.1.4 - Web Services Modell* vorgestellte Web Service Modell mit den entsprechenden, für die einzelnen Bereiche zuständigen, Technologien vervollständigt werden.

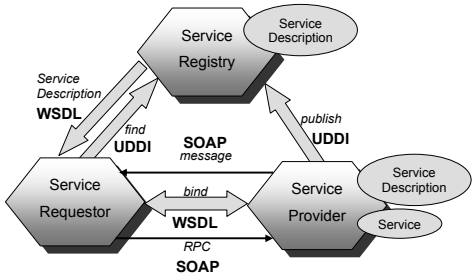

Abbildung 2-25 Web Services Modell und Technologien

2.1.6 Web Services Frameworks

Bisher konnten mit XML nur Informationen auf heterogenen Systeme über das Internet verteilt werden. Durch Web Services ist dies nun auch mit Prozessen möglich. Wenn man einen Web Service entwickeln oder eine bereits bestehende Anwendung Web Service tauglich machen will, erfordert dies mehr als nur die Implementierung des Service selbst. Eine Enterprise-Anwendung muss hochverfügbar und fehlertolerant sein und vielen anderen Anforderungen genügen. Sowohl Microsoft (mit DNA, COM, VB usw.) als auch Sun (mit älteren Versionen von J2EE) hatten erfolgreiche Konzepte für Enterprise-Anwendungen, welche aber nicht entworfen worden waren, um mit Web Services umgehen zu können. Mit jeweils neuen Konzepten repositionieren sich beide Unternehmen und stellen sich den neuen Anforderungen. Microsoft stellte im September 2000 seine .NET Initiative vor und Sun antwortete im Februar mit Sun ONE (Open Network Environment). Beide Frameworks zur Erstellung, Deployment und Administration von Web Services sollen hier kurz vorgestellt werden. Diese Ansätze sind die am weitesten verbreiteten, aber auch andere Global Player und einige Startups, welche sich dadurch schnell einen Namen machen konnten, stellten Frameworks in diesem Bereich vor:

- HP NetAction
- IBM Web Services Architecture / IBM WebSphere
- Oracle .NOW / Oracle 9i
- Bea WebLogic
- Iona ORBIX
- CapeClear CapeConnect
- SilverStream eXtend
- Shinka Business Integration Platform

2.1.6.1 Smart Web Services

Sowohl Sun als auch Microsoft erweitern das Modell der Web Services um Kontextsensitivität, wodurch aus Web Services so genannte ´Smart Web Services´ entstehen. Solche Web Services ´wissen´, von wem, wann, wo, warum und unter welchen Sicherheitsbedingungen sie aufgerufen wurden. Die folgenden Punkte geben eine detaillierte Übersicht, über situationelle Bedingungen, welche einen Smart Web Service charakterisieren:

- die Identität des Service Requestors – ist er ein Individuum, eine Organisation oder ein anderer Web Service?
- die Rolle des Konsumenten zu der Zeit, als der Service aufgerufen wurde
- die Präferenzen, die für diesen Service definiert wurden
- mit dem Benutzer des Services assoziierte Sicherheits-, Privacy- und Geschäftsaspekte
- der physische Standort des Service Requestors
- die früheren Aktionen des Service Requestors (History)
- das Endgerät, welches benutzt wird, um den Web Service anzusprechen
- die Vereinbarungen zwischen dem Service Requestor und dem Service Provider wie zum Beispiel Zahlungsmodalitäten

Je mehr Web Services entstehen und je mehr Kontextinformationen diese sammeln, desto komplexer werden deren Interaktionen. Für den Menschen werden bestimmte Interaktionen so erscheinen, als hätte dieses Netz aus verteilten Diensten eine Art Intelligenz oder Selbst-Bewusstsein. Intelligenz würde in diesem Fall als Netzwerkeffekt auftreten – im Endeffekt ist unser Gehirn selbst nur ein komplexes Netzwerk aus relativ einfachen Neuronen. Beim ersten Mal, wenn eine Kollaboration interagierender Web Services ein sinnvolles, unerwartetes Verhalten zeigt, werden wir denken, dass Intelligenz im Spiel ist. Das folgende (etwas abgeänderte) Beispiel aus [RAWS01] soll dies verdeutlichen: Ein deutscher Geschäftsreisender trifft am Flughafen in New York ein und mietet dort sowohl Auto als auch Handy, da sein Anbieter Amerika nicht abdeckt. Als er zwei Tage später von seinem Meeting zurück zum Flughafen fährt, klingelt sein gemietetes Handy und seine Reiseplaner-Applikation teilt ihm mit, dass sein Flug Verspätung hat. Das erste Mal, wenn dieses Verhalten auftritt, ist man sicher überrascht und mutet der Reiseplaner-Applikation eine gewisse Intelligenz zu, insbesondere wenn diese Fähigkeit vorher nicht explizit festgelegt wurde. Dieses Szenario kann durch die folgende Sequenz einfacher Web Service Interaktionen aufkommen: Als der Reisende Auto und Handy mietete, verlangte die Reiseplaner-Applikation vom Verleihservice nach den Mietverträgen und teilte die Kosten umgehend der firmeninternen Reisekostenverwaltung mit. Beim Erhalten und Aufschlüsseln nach Posten bemerkt die Reisekostenverwaltung das Handy und aktualisiert damit sofort das firmeninterne Telefonverzeichnis. Als später

die Reiseplaner-Applikation die Abflugzeit vom Flughafen einholt und die Verspätung gemeldet bekommt, fragt sie als erstes das firmeninterne Telefonverzeichnis ab, erhält die neue Handynummer und verständigt den Reisenden. Ein anderer Punkt, den dieses Szenario erahnen lässt, ist, dass unerwartete Kollaborationen und Interaktionen unerwünscht sein können. Es müssen in Zukunft entsprechende Sicherheitsvorkehrungen und Kontrollmechanismen festgelegt und erstellt werden, welche uns vor unerwünschten Konsequenzen schützen. Führt man diese Vision fort, entstehen verteilte Dienste, die auch als ´Schwärme´ bezeichnet werden. Diese finden sich gegenseitig und erstellen gemeinsam zum richtigen Zeitpunkt genau den Dienst, den der Benutzer benötigt.

Im Folgenden sollen nun die beiden kontrahierenden Frameworks Sun ONE und .NET kurz vorgestellt werden, welche als Entwicklungsplattform und Infrastruktur für Smart Web Services dienen und in Zukunft dieses Szenario vielleicht Wirklichkeit werden lassen könnten. An dieser Stelle soll auf einen Vergleich oder eine Wertung dieser Frameworks verzichtet werden. Es sei aber auf eine Veröffentlichung der Middleware Company verwiesen, die eine interessante Studie zu diesem Thema veröffentlicht hat[1].

2.1.6.2 Sun ONE (Open Network Environment)

Sun ONE (Open Network Environment) stellt eine offene Architektur bereit, in der Web Services entwickelt, über das Internet zur Verfügung gestellt und administriert werden können. Parallel zu dieser offenen Architektur bietet Sun mit iPlanet ein entsprechendes Produktportfolio an, welches eine Art Referenzimplementierung dieser Architektur darstellt.

Sun teilt Web Services generell in zwei Kategorien: Micro Services und Macro Services. Bei Micro Services handelt es sich um einfache grundlegende Funktionen, durch deren geschickte Kombination Macro Services erstellt werden können. Bestandteile der Architektur, welche in *Abbildung 2-26 Sun ONE Basisarchitektur* zu sehen ist, sind die folgenden:

- Service Platform
  Die Service Platform erlaubt Verbindungen zu Datenbanken, Verzeichnis-

[1] Vgl. [JVSM].

diensten, Messaging-Services usw. Sie basiert auf einem Betriebssystem oder einer virtuellen Maschine und hat Zugriff auf Hardware und Netzwerk.

- Service Container
  Sowohl Micro Services als auch Macro Services werden innerhalb eines Service Containers ausgeführt. Dieser Service Container, welcher zum Beispiel ein Applikationsserver sein kann, der auf einer Service Platform läuft, stellt eine Laufzeitumgebung zur Verfügung, verwaltet Persistenz, Transaktion und Status. Sun ONE empfiehlt auf Serverseite einen J2EE-Applikationsserver und auf der Konsumentenseite die J2SE[1] bzw. J2ME[2] Technologie.
- Service Integration
  Diese Schicht ermöglicht die Anbindung an Backend Systeme, wie Mainframes, ERP-Systeme usw.
- Service Creation & Assembly
  Dieser Bestandteil repräsentiert die Werkzeuge, die benutzt werden, um Systeme, die auf dem Web Services Modell basieren, zu entwickeln. Der Entwicklungsprozess besteht hierbei, laut Sun, aus zwei Schritten. Im ersten Schritt werden diskrete Services (Micro Services) implementiert, welche im zweiten Schritt zu Macro Services assembliert werden. Entwickler erzeugen Micro Services, indem sie IDEs, Code-Generatoren, XML-Editoren und andere Werkzeuge verwenden. Service Assemblierer benutzen Policy-, Modellierungs- und Workflow-Tools. Sind die Services ausreichend getestet, sind sie für das Deployment bereit. Dieses und die Möglichkeit der Überwachung sowie Administration der Services werden durch verschiedene Tools der Architektur ermöglicht.
- Service Delivery (Service Interface)
  Diese Schicht stellt grundlegende Verbindungs-, Standort-, Discovery- und Kommunikations-Funktionalitäten bereit und delegiert jeden Request zum passenden Service.

[1] Die Java 2 Standard Edition (J2SE) ist die Programmier- und Laufzeitumgebung für Desktop-Clients.

[2] Die Java 2 Micro Edition (J2ME) ist die Programmier- und Laufzeitumgebung für mobile Endgeräte.

- Applications / Web Services
  Diese Schicht repräsentiert die bereits deployten Web Services und die dahinter liegenden Applikationen.

Abbildung 2-27 – Sun ONE Smart Web Services Architektur zeigt eine, um Smart Web Services erweiterte, Architektur, welche mehr Funktionalität hinzufügt, um Kontextsensitivität zu unterstützen. Die Architekturerweiterungen sind die folgenden:

- Smart Interface (Smart Delivery)
  Dieser Architekturbestandteil ist für die kontextsensitive und gerätespezifische Auslieferung der Inhalte an die Konsumenten zuständig. Er unterstützt durch Verwendung von XSLT (siehe *Kapitel 2.1.2.3.7 – XSL – Transformation und Darstellung*) gerätespezifische Präsentationsformate wie HTML, XHTML, WML, VoiceXML usw.
- Smart Policy
  Dieser Bereich umfasst die Koordination der Policy-Aktivitäten in Verbindung mit Identität, Kontext und Rollen.
- Smart Process
  Dieser Architekturbestandteil ermöglicht die kontextabhängige Kollaboration und Interaktion von Web Services, d.h. an dieser Stelle wird der Workflow realisiert.
- Smart Management
  Hier wird die Verwaltung und Überwachung der Web Services sowie der Plattform vorgenommen, wobei Privacy, Sicherheit und Rechtevergabe auf einer spezifischen, durch den Kontext definierten, Situation basieren. Sun ONE empfiehlt hier die Java Management Extensions (siehe *Kapitel 2.2.6 – JMX – Die Management-Architektur von Morgen?*).

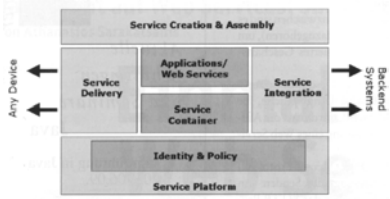

Abbildung 2-26 Sun ONE Basisarchitektur (Quelle: [JM9/01a])

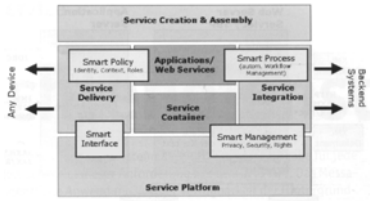

Abbildung 2-27 Sun ONE Smart Web Services Architektur (Quelle: [JM9/01a])

Der typische Ablauf einer Benutzerinteraktion mit dem System sieht folgendermaßen aus: Requests können von jedem beliebigen Gerät eintreffen. Der aktuelle Kontext wird durch die Smart Delivery verarbeitet, Standortinformationen, Rolle und Zeit werden gespeichert und der Request wird an den entsprechenden Web Service delegiert. Der Smart Process Bereich verwaltet die Choreografie der Web Services und garantiert, dass der richtige Micro Web Service in korrekter Reihenfolge des Workflows aufgerufen wird. Die Services selbst verwenden die Integrations- und Ressourcen-Engines, um andere Services aufzurufen und um auf Datenbanken, Dateien, Legacy-Applikationen usw. zuzugreifen. Ist dies erfolgreich geschehen, wird die Antwort für das Endgerät, d.h. den Konsumenten, generiert, wobei sowohl Personalisierung, als auch Kontextsensitivität benutzt werden. Die Antwort wird vom Konsumenten durch den entsprechenden Kanal empfangen.

Wie diese Architektur in der Praxis umgesetzt werden kann, beantworten *Abbildung 2-28 Sun ONE Architektur Standards* und der folgende Absatz. Die in der Abbildung mit ´*´ gekennzeichneten Technologien sind entweder noch im Beta-Stadium oder werden erst in naher Zukunft erscheinen.

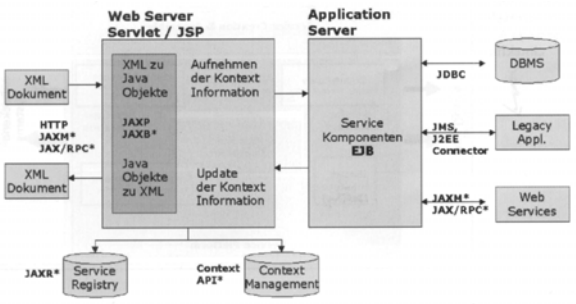

Abbildung 2-28 Sun ONE Architektur Standards (Quelle: [JM9/01a])

Die Sun ONE Architektur basiert auf der Java Plattform und nutzt die bereits bestehende J2EE-Infrastruktur. Es existieren (oder befinden sich noch in Arbeit) eine Reihe von Java XML APIs, welche zu J2EE hinzugefügt werden müssen, um diese Architektur Web Service fähig zu machen:

- JAXP (Java API for XML Processing)
  dient dem Parsen (DOM, SAX) und Verarbeiten (XSL) von XML-Dokumenten.
- JAXM (Java API for XML Messaging)
  dient dem Verarbeiten und Versenden von asynchronen SOAP-Nachrichten.
- JAX/RPC (Java API for XML Remote Procedure Call)
  dient dem Verarbeiten und Versenden von synchronen SOAP-Nachrichten.
- JAXR (Java API for XML Registries)
  dient dem Zugriff auf XML-Registries, wie ebXML Registry und UDDI Registry.
- JAXB (Java API for XML Binding)
  dient zur Abbildung von XML Daten auf Java Objekte und vice versa.

Diese Technologien werden zurzeit als ´JAX Pack´ (Java APIs for XML Pack) ausgeliefert und sollen in das für Mitte des Jahres 2002 angekündigte ´Web Services

Pack´ integriert werden, welches zusätzlich den ´JavaServer Faces´[1] Standard, die Jakarta Tomcat Servlet/JSP Engine[2] und die ´Java API for WSDL´[3] enthält.

Die Umsetzung des Web Services Szenarios mit Hilfe der zur Verfügung stehenden Java APIs stellt sich folgendermaßen dar: Eine Web Service Schnittstelle ist als Servlet oder JSP implementiert. Das Servlet bzw. die JSP empfängt die SOAP-Nachricht und extrahiert das XML-Dokument. Anschließend wird dieses Dokument in ein oder mehrere Java Objekte konvertiert. Danach wird die Geschäftslogik aufgerufen, welche meist in Form eines Enterprise Java Beans (EJB) existiert. Mittels J2EE Connector API können ERP-Systeme wie SAP R/3 oder Mainframes angebunden werden, per JDBC Datenbanken und per SOAP natürlich auch andere Web Services. Das Ergebnis dieser Operationen wird in einem Java Objekt gekapselt, das wiederum in ein XML-Dokument konvertiert und zurückgesendet wird. Ein ähnliches Framework bietet Microsoft .NET, welches im Folgenden vorgestellt werden soll.

2.1.6.3 Microsoft .NET

Über kurz oder lang dürfte Microsoft .NET alle Softwareentwickler betreffen, deren Lösungen auf Microsoft-Technologien beruhen, da dieses Framework alle bisher nebeneinander existierenden Bereiche umfasst und integriert. Die .NET Strategie hält dabei Einzug in die komplette Produktlandschaft von Microsoft, angefangen von Betriebssystemen bis hin zu Backoffice-Lösungen. Wie auch Sun ONE dient .NET in erster Linie der Entwicklung von Web Services und als Laufzeitumgebung für diese. Der Begriff ´.NET Platform´ steht nicht nur für eine neue Technologie oder ein neues Framework, sondern, im Gegensatz zu Sun ONE, für eine ganze Produktpalette und Ausrichtung von Microsoft. Microsoft definiert .NET folgendermaßen: „.NET is our strategy for delivering on the Microsoft vision: Empowering people through great software, any time, any place, on any device“. Die folgende Abbildung zeigt die Gesamtsicht auf dieses umfassende Framework, welches im Folgenden erläutert wird.

---

[1] Hierbei handelt es sich um ein auf dem JavaServer Pages Standard aufbauendes Framework, womit sich auf einfache Art und Weise HTML Client GUIs erstellen lassen. Es gleicht eine Art Java Swing für HTML Browser.

[2] Weitere Informationen in *Kapitel 3.4.3 – Technologien, Produkte und APIs*.

[3] Durch diese API wird der Zugriff auf und die Arbeit mit WSDL-Dokumenten ermöglicht.

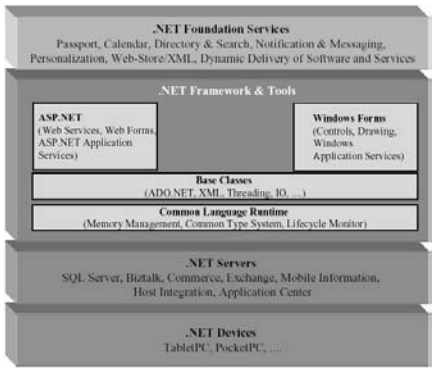

Abbildung 2-29 .NET Architektur (Quelle: [MSNET], S.13)

- .NET Foundation Services
  Hierbei handelt es sich um von Microsoft betriebene, endbenutzerorientierte Web Services. Unter dem Namen ´Hailstorm´ soll eine Grundmenge dieser Dienste erhältlich sein, welche es erlauben, Informationen zentral im Internet zwischen verschiedenen Services zu teilen. Zu den derzeit verfügbaren Diensten zählen:
  - myAdress – elektronische und geografische Adresse einer Identität
  - myProfil – Name, Nickname, spezielle Daten und Bild
  - myContacts – Kontaktverwaltung
  - myLocation – elektronischer und geografischer Aufenthalt
  - myInbox – E-Mail und VoiceMail Eingangskorb
  - myCalender – Zeit- und Aufgabenverwaltung
  - myWallet – Ablage für Quittungen, Zahlungsanweisungen usw.
  - myNotification – Benachrichtigungsdienst
  - myApplicationSettings – benutzerspezifische Applikationseinstellungen
  - myFavoriteWebSites – Favoritenverwaltung
  - myDevice – Geräteeinstellungen und –fähigkeiten
  - myDocument – Dokumentenablage
  - myServices – Verwaltung verschiedener Dienste
  - myUsage – Verwendungs-Reports für verschiedene Dienste

- .NET Framework & Tools
  Das .NET Framework ist das Herz von .NET. Es enthält die Laufzeitumgebung ´Common Language Runtime´ (CLR), verschiedene Compiler und die umfassende Entwicklungsumgebung ´Visual Studio .NET´. Auf diesen Hauptbestandteil der Architektur wird weiter unten näher eingegangen.
- .NET Servers
  Hierbei handelt es sich um eine Reihe von Server-Produkten, welche die Basis für das .NET Framework bilden. Folgende Enterprise-Server sind in dieser Produktpalette enthalten:
  - Biztalk Server 2000 – Geschäftsprozess- und Integrationsserver
  - SQL Server 2000 – Datenbankverwaltungssystem
  - Exchange Server 2000 – Messaging-Server
  - Host Integration Server 2000 – Anbindung an Host-Systeme
  - Commerce Server 2000 – E-Commerce Server
  - Mobile Information Server 2000 – Applikationsserver mit Mobile-Anbindung
  - Internet Security and Acceleration Server 2000 – Web-Server
  - Content Management Server – Content Management
  - SharePoint Portal Server – Intranet Server
  - Windows 2000 Server – Directory-, Web-, File-, Print-Server
  - Application Center 2000 – Deployment- und Management-Umgebung
- .NET Devices
  Dieser Bestandteil dient zur Unterstützung einer breiten Palette von Endgeräten wie PCs, Laptops, Handys, PDAs, Spielkonsolen usw. Vorgesehene Laufzeitumgebungen sind das .NET Framework und das .NET Compact Framework, welches eine Untermenge für mobile Endgeräte darstellt.

Den Kern der .NET Plattform bildet, wie bereits angesprochen, das ´.NET Framework´, welches aus den folgenden Komponenten besteht:

- Common Language Runtime (CLR)
  Hinter CLR verbirgt sich eine objektorientierte Laufzeitumgebung, mit der sich Code aus beliebigen Programmiersprachen (außer Java) integrieren lässt. Zu diesem Zweck übersetzen CLR-kompatible Entwicklungsumgebungen die Programme zunächst in den CPU-unabhängigen Zwischencode MSIL (Microsoft Intermediate Language) für eine theoretische virtuelle Maschine. Die Intermediate-Language-Files werden in Deploymenteinheiten, so genannten

´Assemblies´, verpackt. Erst zur Laufzeit erfolgt mit Hilfe eines Just-in-Time-Compilers die Übersetzung in den jeweiligen nativen maschinenabhängigen Code. Dadurch erreicht .NET eine Entkopplung zwischen Anwendungssoftware und der darunter liegenden Systemumgebung.

- Base Classes
  Hierbei handelt es sich um eine umfangreiche Klassenbibliothek, mit der nahezu alle Bereiche der Programmierung abgedeckt werden. Diese Klassenbibliothek steht allen CLR-kompatiblen Programmiersprachen zur Verfügung. Wichtigste Bestandteile sind Threading, I/O, XML-Unterstützung und ADO.NET (ActiveX Data Objects), eine datenbankunabhängige Datenbankzugriffsschicht.
- ASP.NET
  Active Server Pages ist Microsofts Modell zur Programmierung der Darstellungsebene, d.h. ASP ermöglicht die Generierung dynamischer Seiten (HTML, WML usw.).
- Windows Forms
  Die Windows Forms bzw. WebForms Technologie erlaubt die Integration von Benutzer-Kontrollelementen auf Webseiten und deren Verbindung zu serverseitigem Code. Dadurch ist es dem Entwickler möglich, Web-Seiten visuell so zu konstruieren, wie dies bisher nur bei Visual Basic möglich war.

2.1.7 Auswirkungen auf Geschäftsprozesse und –modelle

"Businesses that ignore its potential, or decide to sit out its early stages, will find themselves outpaced by rivals that take advantage of Web Services to improve their agility and even to transform themselves into new kinds of enterprises."[1]

Mit der Etablierung der Web Services Technologie wird mit einer weitreichenden Veränderung der Geschäftsprozesse gerechnet. Nach Gartner[2] ist sogar mit dramatischen Auswirkungen auf die Geschäftswelt zu rechnen, weshalb folgende Voraussagen gemacht werden:

- Im zweiten Halbjahr 2002 werden 75% der Unternehmen mit mehr als 100 Millionen Dollar Umsatz teilweise durch Web Services verbunden sein. (Aussagenwahrscheinlichkeit bei 80%)
- Im ersten Halbjahr 2003 werden 50% der Unternehmen mit weniger als 100 Millionen teilweise durch Web Services verbunden sein. (Aussagenwahrscheinlichkeit bei 80%)

Die Form der Wertschöpfungskette eines Unternehmens hat sich seit den achtziger Jahren durch Einflüsse wie ´Supply Chain Management´, ´Outsourcing´ usw. in der in *Abbildung 2-30 Evolution der Wertschöpfungskette* dargestellten Art und Weise verändert. Durch Technologien und Architekturen, wie der ´Web Service´ Architektur, werden weitere evolutionäre Veränderungen erwartet.

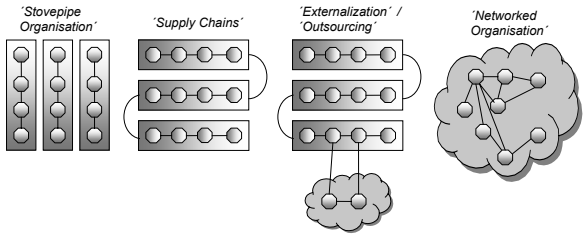

Abbildung 2-30 Evolution der Wertschöpfungskette

[1] Vgl. [WSGART], S.1.

[2] Vgl. [WSGART], S.2.

Die ursprünglichen Organisationen, welche in [CBDIWS] als ´stovepipe organisations´ bezeichnet werden, verfügen über eine klare Trennung ihrer Geschäftsbereiche und deren Wertschöpfungsketten. Die Integration der jeweils zuständigen Applikationen ging nicht über das Papier, das zwischen den einzelnen Organisationseinheiten ausgetauscht wurde, hinaus. Später lag der Fokus auf der Automatisierung und Kopplung der Wertschöpfungsketten, doch obwohl hier Client/Server Architekturen verwendet werden sind die Geschäftslogikprozesse durch die physikalische Trennung in diesen Applikationen meist stark gekoppelt, so dass es sehr schwierig ist, Teile des Systems auszutauschen oder wieder zu verwenden. In den letzten Jahren lagerten immer mehr Unternehmen bestimmte Prozesse an darauf spezialisierte Unternehmen aus, um sich auf ihre Kernkompetenzen konzentrieren zu können. Allerdings war auch hier oft keine wirkliche Integration zwischen den Systemen der Partnerunternehmen vorhanden, da die Daten in den meisten Fällen dupliziert und konvertiert wurden. Das ´Web Service Paradigma´ erlaubt eine unkomplizierte Integration der Geschäftsprozesse verschiedener Unternehmen in ein Prozessnetzwerk. Diese neue Form des Unternehmens wird als ´virtuelles Unternehmen´ bezeichnet, welches sehr schnell und dynamisch auf Änderungen reagieren kann, um seine Effizienz zu erhöhen. Dazu gehören auch so genannte ´hot plug and play partnerships´, d.h. die dynamische Integration neuer Partner zu jedem beliebigen Zeitpunkt.

2.1.8 Zusammenfassung und Ausblick

Das Internet begann als ´Document Web´, welches Wissenschaftlern diente, Forschungsdokumente auszutauschen. Es bestand aus Web Servern, welche mittels HTTP kommunizierten und die Informationen in HTML lieferten, um sie im Browser darzustellen. Dann bemerkten die Unternehmen, dass dieses Document Web auch für Unternehmenszwecke geeignet ist und es entstanden viele Web-Seiten, um Produkte und Dienste anzubieten. Als nächsten Schritt wollten Unternehmen Transaktionen mit ihren Kunden über das Internet abwickeln und so entstand das ´application web´. Dieses besteht aus Applikationsservern, die dynamische HTML- oder WML-Seiten aus serverseitiger Geschäftslogik erzeugen, was den Beginn des E-Commerce darstellte. Nun wollen die Unternehmen ihre Prozesse über das Internet mit anderen Unternehmen integrieren und es entsteht das ´Service Web´. Dieses besteht aus Web Application Servern, welche mittels SOAP kommunizieren und die Informationen in XML austauschen.

Das Internet bietet eine unüberschaubare Menge an Informationen, seien dies nun Aktienkurse, Flug- und Reiseinformationen oder E-Shops. Man kann diese Informationen als einfache präsentierte HTML-Seiten betrachten, aber es existiert noch eine andere Sichtweise: Diese Seiten können als Dienst bzw. Web Service betrachtet werden, welcher über HTTP angesprochen wird und seine Informationen in der Sprache HTML liefert. Warum muss jeder Aktieninformationsanbieter ein eigenes Modul für die Darstellung der Kurse in seine Seite integrieren, obwohl ein gemeinsames optimiertes Modul genügen würde? Oder warum sollte nicht jeder die aktuellen Aktienkurse oder zum Beispiel ein Übersetzungsmodul auf seiner Homepage zur Verfügung stellen können? Diese Idee der Web Services soll eine neue Welt des Internets ermöglichen: Monolithische Systeme und auf einzelnen PCs installierte Software verschwinden und werden durch verteilte Systeme auf Basis von Web Services ersetzt. Dies bedeutet, dass es nur noch ´einen Computer´ geben wird: Dieser Computer ist das Internet und auf diesem Computer arbeiten verteilte Dienste. Diese verteilten Dienste werden auch als ´Schwärme´ bezeichnet, welche sich gegenseitig finden und gemeinsam genau den Dienst erstellen, den der Benutzer benötigt und dies in genau dem Augenblick, in dem er ihn braucht. Um das zu ermöglichen, sollen Technologien wie JINI[1] oder ´Universal Plug and Play´[2] zum Einsatz kommen, die spontane Zusammenschlüsse bzw. die spontane Vernetzung von Web Services ermöglichen. Es wird im Bereich der Web Services in den nächsten Jahren sicher noch viele Entwicklungen geben müssen, um dieser Vision näher zu kommen.

[1] Diese Technologie und Dienstinfrastruktur ermöglicht das spontane Auffinden und Interagieren von Ressourcen in einem Netz. Weitere Informationen unter http://www.sun.com/jini.

[2] Diese von Microsoft propagierte Technologie ermöglicht den Betrieb von P2P-Netzen. Weitere Informationen finden sich unter http://www.microsoft.com/hwdev/tech/nonpc/UPnP.

2.2 Management-Architekturen

Eine alltägliche Situation: Bei der Arbeit mit Windows bemerkt man, dass die CPU-Auslastung ohne ersichtlichen Grund 100% beträgt. Daraufhin versucht man folgendes:

- öffnen des Task-Managers um den CPU-Verbrauch der Prozesse zu betrachten und gegebenenfalls einen dieser Prozesse zu beenden
- analysieren der Systemeinstellungen
- untersuchen der Festplattenaktivität und des Paging-Verhaltens

Nun stelle man sich ein Produktivsystem vor, welches:

- auf fünf gekoppelten Servern läuft,
- einen Apache-Server[1] für die Verarbeitung der Frontend-Aktivitäten hat,
- einen BEA WebLogic Applikationsserver[2] für die Geschäftslogik benutzt,
- seine Daten zentral mit Hilfe des OODBMS ObjectStore[3] speichert,
- auf Legacy-Daten[4] über ein Tuxedo-basiertes Gateway[5] zugreift
- und natürlich eine 24x7-Verfügbarkeit haben soll.

In diesem Fall sollte Folgendes gewährleistet sein:

- Der Zustand des Systems sollte von einem zentralen Punkt überwachbar sein.
- Das System sollte über problematische Situationen automatisch informieren.
- Die Quelle eines Problems sollte schnell und effizient auffindbar sein.
- Es muss möglich sein, bestimmte Probleme zu beheben, ohne das System herunterfahren zu müssen.
- Der Zugriff auf Ressourcen sollte geregelt und überwacht ablaufen.

[1] Der zurzeit am weitesten verbreitete Web-Server. Weitere Informationen unter http://apache.org.

[2] Der derzeitige Marktführer unter den Applikationsservern. Weitere Informationen unter http://www.bea.com.

[3] Der derzeitige Marktführer im Bereich der OODBMS. Weitere Informationen unter http://www.objectdesign.com .

[4] Unter Legacy-Daten versteht man im Allgemeinen Datenbestände aus der Zeit vor dem Internet-Zeitalter.

[5] Tuxedo ist ein hochperformanter TP-Monitor von Bea Systems. Weitere Informationen unter http://www.bea.com.

Dieses Beispiel zeigt eindrucksvoll die enorme Bedeutung, welche, insbesondere in der heutigen Zeit, einem System zukommt, das für die Überwachung bzw. das Management von Ressourcen aller Art zuständig ist.

Netze, Systeme und Anwendungen werden immer komplexer, die Anzahl der Benutzer, die bei ihrer Arbeit von diesen abhängig sind, nimmt stetig zu. Die Einbindung in heterogene Netze, wie das Internet, sowie die zunehmende Anzahl komplexer verteilter Systeme verstärken diesen Aspekt noch. Das sichere Funktionieren und die schnelle Anpassbarkeit an neue Anforderungen bekommen zunehmend unternehmenskritische Bedeutung. Weiterhin ist eine manuelle Konfiguration und Überwachung angesichts der enormen Komplexität nicht mehr möglich. Es ist daher unabdingbar, geeignete Managementsysteme einzusetzen, welche Aspekte der Planung, Überwachung, Kontrolle sowie Fehlererkennung übernehmen.

2.2.1 Begriffe, Anforderungen, Historie und Entwicklungen

2.2.1.1 Der Begriff ´Management´

"Movement through the jungle of management technologies is an intricate and complex journey, fraught with peril for the uninitiated."[1]

Der Begriff ´Management´ ist sehr breit gefächert und wird je nach verwendetem Kontext anders assoziiert. Management ist aber nicht nur die Überwachung von physikalischen Netzwerkkomponenten wie Routern, Hubs und Switches, sondern auch die Verwaltung und Überwachung von Ressourcen aller Art, welche durch ein Netzwerk miteinander verbunden sind. Dies umfasst ebenso die softwaretechnische Seite, d.h. die Überwachung von Diensten, Applikationen oder komplexen ERP-Systemen sowie deren Komponenten. Management wird in erster Linie durch Menschen (Netzwerkadministratoren, J2EE-Administratoren, Operatoren usw.) mit Hilfe von speziellen Werkzeugen (Hard- und Softwarekomponenten) realisiert. Es handelt sich hierbei also zunächst einmal um ein organisatorisches Problem. Das bedeutet, dass entsprechende Richtlinien und Prozeduren erarbeitet werden müssen, die einen kosteneffektiven und flexiblen Betrieb des Netzes ermöglichen. Hierbei sind Werkzeuge und deren technologische Grundlagen lediglich Hilfsmittel zum erfolgreichen Management.

Man kann Management, wie in *Abbildung 2-31 Klassifikation des Managements* dargestellt, nach den zu überwachenden Ressourcen klassifizieren:

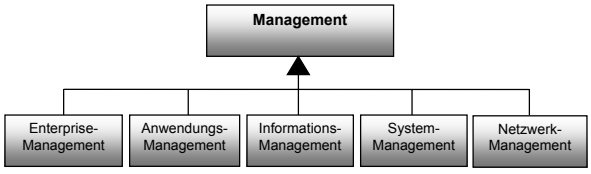

Abbildung 2-31 Klassifikation des Managements

[1] Vgl. [JAVAW2].

- Enterprise-Management
  Das Enterprise-Management hat zur Aufgabe, die Teilbereiche des Managements im Kontext eines Unternehmens zu sehen. Dies bedeutet im Enterprise-Management werden die Aufgaben der einzelnen Unternehmensbereiche zusammengefasst und daraus Zielvorgaben für die IT-Infrastruktur abgeleitet.
- Anwendungsmanagement
  Es wurde deutlich, dass auch verteilte Anwendungen und Dienste Managementaufgaben beinhalten, die von einem Managementsystem gelöst werden müssen. Beispiele hierfür sind Steuerungssysteme zur automatischen Fertigung, verteilt realisierte Informations- und Auskunftssysteme, E-Commerce-Systeme und die immer mehr an Bedeutung gewinnenden Systeme zur Unterstützung von Gruppenarbeit. Zu dieser Form des Managements zählen auch Installation und Deinstallation von Anwendungen, sowie globale und benutzerspezifische Einstellungen auf Anwendungsebene.
- Informationsmanagement
  Diese Form des Managements umfasst alle Managementtätigkeiten in Bezug auf unternehmensweite Datenbestände, deren Entwurf und Pflege, sowie konsistente Haltung. Das heißt hier findet eine Überwachung der Datenbanken, Repositories, Backup-Systeme usw. statt.
- Systemmanagement
  Im Systemmanagement werden nicht, wie im Bereich des Netzwerkmanagements, Netzkomponenten, sondern die Endsysteme der Rechnernetze verwaltet. Die Aufgabe des Systemmanagements besteht also darin, die verteilt erbrachten Systemfunktionen so zu organisieren, dass vernetzte Einzelsysteme nach außen hin wie ein (virtuelles) Gesamtsystem erscheinen.
- Netzwerkmanagement
  Diese Form des Managements befasst sich mit der Verwaltung von Netzdiensten und Netzkomponenten, d.h. der untersten Ebene eines Netzwerkes. Dazu gehören Komponenten wie Router, Switches, Hubs, Bridges, Repeater, aber auch die Leitungen, über die die Kommunikation stattfindet. Es wird daher manchmal auch als Komponenten-Management bezeichnet.

2.2.1.2 Dimensionen des Managements

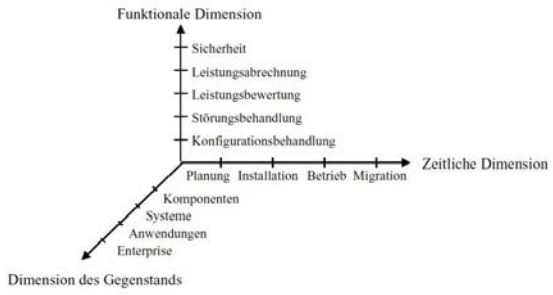

Abbildung 2-32 Dimensionen des Managements (Quelle: [SJM01], S.8)

Abbildung 2-23 Dimensionen des Managements zeigt die Dimensionen und die damit verbundene Komplexität des Managements auf.

Die funktionale Dimension fasst anfallende Aufgaben beim Betrieb eines Netzes und seiner Ressourcen zu Aufgaben zusammen. Man spricht hier auch von so genannten Funktionsbereichen. Die zeitliche Dimension liefert eine Einordnung in den Lebenszyklus eines Netzwerkes, also von der Planung über die Installation bis hin zum Produktivbetrieb. Unter Migration versteht man in diesem Zusammenhang den Umstieg von einer Netzwerktechnologie auf die nächste. Die Dimension des Gegenstandes zeigt die Art der zu verwaltenden Ressourcen auf, d.h. das Management kann sich auf einzelne Komponenten im Netz, auf komplexe Systeme, auf (verteilte) Anwendungen oder die gesamte Netzwerk-Infrastruktur eines Unternehmens beziehen. Das ausschlaggebende Merkmal ist hierbei die Menge der für eine Aufgabe benötigten Informationen.

2.2.1.3 Anforderungen an Managementsysteme

An Managementsysteme wird eine Reihe von Anforderungen gestellt, die gleichzeitig Ziele der aktuellen Entwicklungen in diesem Bereich verkörpern. Diese Anforderungen sollen hier kurz dargestellt werden:

- Sicherstellung der Funktionsbereitschaft eines Netzes
  Dies betrifft Punkte wie die Aufrechterhaltung der Serviceleistungen trotz technologischer Änderungen und hoher Wachstumsraten, die Sicherung der Dienstgüte (z.B. Antwortzeiten) und vorbeugende Fehler- und Engpassbehebung.
- Automatische oder halbautomatische Reaktion auf Betriebsstörungen
  Im Fehlerfall sollte es möglich sein, eine Konfigurationsänderung in Echtzeit vorzunehmen und zum Beispiel redundante Komponenten zu aktivieren, die den aufgetretenen Fehler abfangen.
- Dynamische Reaktion auf Änderungen im Netz und in der Umgebung
  Hiermit sind Änderungen bezüglich Anwendungen, Teilnehmern, Komponenten, Diensten oder Gebühren gemeint, auf die sich ein Managementsystem dynamisch einstellen können sollte.
- Beherrschbarkeit des Netzes
  Relevante Informationen sollten geordnet und komprimiert dargestellt werden. Die Steuerung sollte sowohl zentral als auch dezentral möglich sein. Weiterhin ist eine Integration in heterogene Umgebungen heutzutage unabdingbar.
- Maximale Skalierbarkeit
  Ein weiteres wichtiges Kriterium ist die Skalierbarkeit in Bezug auf die Größe des Netzes, die Zahl der Komponenten und die steigenden Anforderungen an das Management.
- Verbesserung der Arbeitsbedingungen der Administratoren
  Hierzu zählen die Verbesserung und Vereinheitlichung der Benutzeroberflächen von Management-Werkzeugen, die problemlose Integration dieser Werkzeuge in Arbeitsabläufe und die Möglichkeit der schrittweisen Automatisierung von Managementaufgaben.
- Fortschritte durch Standardisierung
  Gerade im Bereich des Netzwerkmanagements existieren eine Reihe von proprietären Lösungen, die einer herstellerunabhängigen Standardisierung

bedürfen. Besondere Bedeutung haben hierbei sowohl offene Schnittstellen zur existierenden Infrastruktur als auch die Integration in bereits bestehende Arbeitsabläufe.

- Maximale Sicherheit
  Gerade im Internet-Zeitalter kommt der Sicherung des Managements gegen Angriffe und Zugriffe von unberechtigten Personen eine besondere Bedeutung zu.

- Multifunktionssysteme
  Der Wunsch nach einheitlichen integrierten Gesamtsystemen, die den gesamten Anforderungsbereich abdecken und für das Management von heterogenen Netzen und Systemen konzipiert sind, wird immer stärker. (siehe hierzu auch *Kapitel 2.2.7 – Integrierte Managementsysteme*)

2.2.1.4 Gegenwärtige Situation

Die folgende Studie der INS[1] von 1999, bei der 200 Unternehmen über die von ihnen eingesetzten Managementsysteme befragt wurden, stellt die gegenwärtige Situation sehr schön dar.

Zufriedenheit mit den verfügbaren Management-Werkzeugen:		Anzahl der eingesetzten Werkzeuge:	
sehr zufrieden	3 %	1 Werkzeug	10 %
zufrieden	28 %	2-5 Werkzeuge	28 %
neutral	34 %	6-10 Werkzeuge	29 %
unzufrieden	30 %	11-20 Werkzeuge	19 %
sehr unzufrieden	5 %	21-35 Werkzeuge	7 %
		mehr als 35 Werkzeuge	7 %
Bedeutung eines verbesserten Netzwerk-Managements:		Hauptprobleme:	
unwichtig	1 %	Qualifiziertes Personal	63.9 %
gering	3 %	Projekte höherer Priorität	57.4 %
wichtig	18 %	Fehlende Prozesse	50.8 %
sehr wichtig	51 %	Fehlende Werkzeuge	49.2 %
äußerst wichtig	27 %	Kosten der Werkzeuge	45.1 %
		Bewilligung der Kosten	41.0 %
		Fehlende Metriken	32.0 %
		Personalwechsel	23.0 %
		Fehlende Dienstverträge	20.5 %
		Andere Probleme	15.4 %

Abbildung 2-33 INS Umfrage 1999 (Quelle: [SJM01],S.7)

[1] Die INS (International Network Services) ist einer der größten Netzwerkspezialisten und wurde Ende 1999 von Lucent Technologies aufgekauft.

Wie man sieht, sind lediglich 31% der befragten Unternehmen mit dem von ihnen eingesetztem Management-System zufrieden. Die Bedeutung die einem verbesserten Management zukommt, ist enorm: für 18% der Befragten ist diese zumindest wichtig, für 51% sehr wichtig und für 27% sogar äußerst wichtig. Die Anzahl der in Kombination eingesetzten Werkzeuge ist erstaunlich hoch, was darauf hinweist, dass integrierte Gesamtsysteme, die den gesamten Anforderungsbereich abdecken, gefragt sind. Das größte Problem im Zusammenhang mit Managementsystemen ist qualifiziertes Personal. Diese Tatsache deutet darauf hin, dass viele dieser Systeme zu komplex oder nicht intuitiv bedienbar sind. In der Tat ist die Bedienerfreundlichkeit oft auf relativ niedrigem Niveau, denn die Bildschirmausgaben der Werkzeuge müssen in vielen Fällen durch das Know-how eines Experten ergänzt werden. Weitere Mängel derzeitiger Systeme, die aus dieser Studie nicht direkt abgeleitet werden können, sind Probleme bei der Fehlerdiagnose, mangelhafte Schwachstellenanalysen und oftmals unzureichende Statistikfunktionen.

2.2.1.5 Standards und Entwicklungen

Bei der Betrachtung des Themengebiets ´Management´ in Bezug auf seine technologische Basis, lassen sich drei grundsätzliche Entwicklungstendenzen entdecken:

- Der OSI-Standard (Open Systems Interconnect), der sehr mächtig und theoretisch fundiert ist, aber aufgrund seiner Komplexität noch wenig Verbreitung gefunden hat.
- Der Standard der TCP/IP-Welt, das Simple Network Management Protocol (SNMP), welcher bereits ein weites Einsatzfeld vorweisen kann, aber noch gewisse konzeptionelle Schwachstellen in sich birgt.
- Neuere Ansätze, wie CORBA-basierte Lösungen, WBEM und die Java Management Extensions (JMX), die flexible, dynamische, skalierbare, integrierte und web-fähige Managementsysteme versprechen.

Auf Historie bzw. die wichtigsten Stationen dieser Management-Technologien soll nachfolgend eingegangen werden.

Jahr	Technologie	Beschreibung
1980	OSI-Standard (CMIP)	ISO startet Aktivitäten im Bereich Netzwerkmanagement als Teil der Entwicklung des ISO/OSI-Referenzmodells. CMIP (Common Management Information Protocol) bezeichnet das hierbei verwendete Protokoll.
1985	TMN	CCITT (heute ITU) beginnt die Definition des Telecommunication Management Network (TMN)
1987	OSI-Standard (CMIP)	Arbeiten zum OSI Systems Management Overview beginnen
1988	OSI-Standard (CMIP)	Gründung des OSI Network Management Forums (NMF), einer Herstellervereinigung zur Umsetzung des OSI-Standards
1988	CMOT	Common Management Information Services over TCP/IP (CMOT) entstand als OSI-Antwort auf SGMP
1988-1992	TMN, OSI-Standard (CMIP)	Derartige Anpassung der OSI-Standards und der TMN-Standards, dass TMN auf den OSI-Standards aufbaut
1991	OSI-Standard (CMIP)	OSI Systems Management Overview wird internationaler Standard
1990-1997	OSI-Standard (CMIP)	Erarbeitung verschiedener Standards, die Managementfunktionen definieren

Tabelle 2-2 Historie des Managements im OSI-Bereich

Jahr	Technologie	Beschreibung
1987	SGMP	Simple Gateway Monitoring Protocol (SGMP) der Firma Proteon und anderer Netzwerkbetreiber.
1987	HEMS	High-Level Entity Management System (HEMS)
1988	SNMPv1	Simple Network Management Protocol Version 1 (SNMPv1) wird von der IETF als Standard verabschiedet.
1993	SNMPv2	Simple Network Management Protocol Version 2 (SNMPv2) wird der IETF vorgelegt, aber nicht standardisiert.
1999	SNMPv3	Simple Network Management Protocol Version 2 (SNMPv2) liegt als ´Draft Standard´ vor

Tabelle 2-3 Historie des Managements im SNMP-Bereich

Jahr	Technologie	Beschreibung
Seit 1995	CORBA-Management	Verschiedene CORBA-basierte Managementansätze
1995	JMAPI	Java Management API (JMAPI)
1999	JMX	Java Management Extensions (JMX)
1999	WBEM	Web-based Enterprise Management (WBEM)

Tabelle 2-4 Neuere Ansätze im Management-Bereich

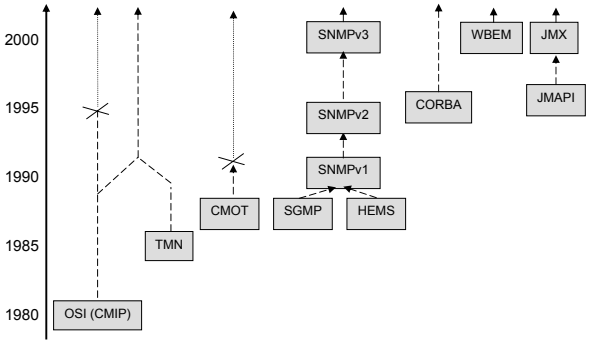

Abbildung 2-34 Historie verschiedener Management-Technologien

1980 startete die ISO erste Aktivitäten im Bereich Management als Teil der Entwicklung des ISO/OSI-Referenzmodells, welches 1984 inklusive des Common Management Information Protocol (CMIP) veröffentlicht wurde. Einen kurzen Abriss über das OSI-Management bietet *Kapitel 2.2.3 – OSI-Management / CMIP*.

Als 1983 die TCP/IP-Protokollfamilie vom Amerikanischen Verteidigungsministerium zum Standard erklärt wurde, war dies die Geburtsstunde des Internets. Mitte der 80er Jahre wuchs das Internet sprunghaft ohne jegliche TCP/IP-Verwaltungsstandards an. Jede verantwortliche Gruppe, die einen Teil des Internets verwaltete, benutzte verschiedene Werkzeuge, Einrichtungen und Verfahren. Ende der 80er Jahre erforschten mehrere unabhängige Gruppen Management-Protokolle. Das früheste Modell war das High-Level Entity Management System (HEMS), doch obwohl es in Versuchs-Netzwerken volle Funktionalität bewies, wurde es im Internet nie angewandt. Wenig später wurde ein weiteres Modell durch die OSI-Gruppe vorgestellt. Es handelte sich hierbei um eine TCP/IP-fähige CMIP-Version, dem so genannten CMIP over TCP/IP (CMOT). Anfang 1987 begann eine andere Gruppe am Simple Gateway Monitoring Protocol (SGMP) zu arbeiten. SGMP war sehr einfach aufgebaut und mühelos zu integrieren, so dass es Ende des Jahres bereits in

mehreren Betriebssystemen implementiert war. Im Februar 1988 versammelten sich schließlich die Mitglieder des Internet Activities Board (IAB)[1], um darüber zu entscheiden, welches der drei Management-Protokolle in Zukunft für das Internet genutzt werden sollte. Die Entscheidung fiel folgendermaßen aus: wegen seiner Akzeptanz und Verbreitung sollte SGMP kurzfristig die erste Wahl sein und langfristig auf CMOT, und damit dem OSI-Standard, umgestellt werden. Um einen zukünftigen Übergang diverser Systeme von SGMP nach CMOT zu gewährleisten, musste ein Internet Management Framework entwickelt werden, dass von beiden Modellen benutzt werden konnte. Dieses Framework wurde Simple Network Management Protocol (SNMP) genannt. Während dieser Zeit wuchsen die Probleme und Unstimmigkeiten zwischen CMOT- und SNMP-Vereinigungen, so dass Mitte 1989 das IAB die Idee des gemeinsamen Frameworks aufgab und eine ´freiheitliche Weiterentwicklung´ beider Ansätze beschloss. Aufgrund der weiten Verbreitung und Stabilität von SNMP wurde es Ende des Jahres als der Management-Standard der Internet-Welt empfohlen. CMOT konnte sich aus verschiedenen Gründen nicht durchsetzen und wird heute nur noch sehr spärlich genutzt. Seine Komplexität und die damit verbundenen Kosten der Implementierung haben dazu geführt, dass auch IBM als letzter sein CMIP-Engagement einstellte und das CMIP/CMOT-Konzept größtenteils vom Markt verdrängt wurde.

SNMP verbreitete sich rasant, wird permanent weiterentwickelt und ist auch heute noch der Standard für das Management von TCP/IP-Netzen. Einen Überblick über das Internet-Management mit SNMP bietet *Kapitel 2.2.4 – Internet-Management / SNMP*.

Zu den neueren Ansätzen zählt das Management mit Hilfe von CORBA. Hierbei existieren verschiedene Ansätze, unter denen JIDM (Joint Inter-Domain Management)[2] am aussichtsreichsten scheint. Ein anderer Ansatz ist die WBEM-Initiative (Web-based Enterprise Management)[3] der DMTF[4]. Dieser versucht hierbei mit Hilfe des Common Information Models (CIM) ein übergreifendes Schema für

[1] Das Internet Activities Board (heute Internet Architecture Board) verfolgt und betreut die Weiterentwicklung der Internet-Standards. Es besteht aus der IETF (Internet Engineering Task Force) und der IRTF (Internet Research Task Force).

[2] Weitere Informationen unter http://www.jidm.org.

[3] Weitere Informationen unter http://www.dmtf.org/standards/standard_wbem.php.

[4] Die Distributed Management Task Force (DMTF) ist eine Herstellervereinigung, welche sich mit der Entwicklung und Standardisierung von Management-Standards beschäftigt. Weitere Informationen unter http://www.dmtf.org.

Informationen über überwachte Objekte (z.B. Drucker, Applikationen, Betriebssysteme) zu definieren und diese Informationen über XML und HTTP verfügbar zu machen. Eine Erläuterung der verschiedenen Facetten des Managements mit Hilfe von CORBA und dem WBEM Ansatz ist nicht Gegenstand dieser Arbeit. Der dritte der neueren Ansätze, welcher insbesondere von Sun vorangetrieben wird, und ein wichtiger Bestandteil dieser Arbeit ist, sind die Java Management Extensions (JMX), in die *Kapitel 2.2.6 JMX – Die Management-Architektur von morgen?* einen Einblick bietet.

2.2.2 Terminologie und grundlegende Konzepte

2.2.2.1 Teilmodelle einer Management-Architektur

Eine Management-Architektur besteht aus vier Teilmodellen, von denen jedes einen bestimmten Aspekt eines herstellerübergreifenden Managementsystems behandelt:

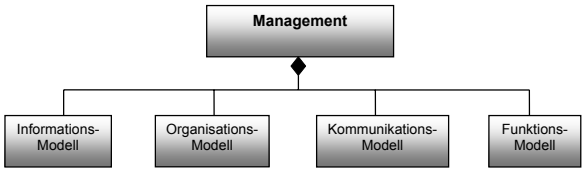

Abbildung 2-35 Teilmodelle einer Management-Architektur

- Informationsmodell
  Im Informationsmodell wird die Syntax der auszutauschenden Managementinformation festgelegt, d.h. es werden zum Beispiel die Regeln zur Beschreibung der zu überwachenden Objekte festgelegt. Dazu gehören Regeln zur Identifikation, Zusammensetzung, Verhalten und Manipulation der Objekte.
- Organisationsmodell
  Das Organisationsmodell ist für die Rollenverteilung einzelner Management-Komponenten verantwortlich, d.h. es enthält Regeln zur Beschreibung der Organisationsstruktur einer Managementanwendung. Ebenfalls im Organisations-Modell spezifiziert werden eventuelle Aufteilungen der Ressourcen in Domänen und eine Zuordnung von so genannten Policies zu allen Ressourcen in bestimmten Domänen.
- Kommunikationsmodell
  Eine Managementplattform ist im Prinzip nichts anderes, als eine verteilte Anwendung, die das Management von Ressourcen zum Ziel hat. Zur Steuerung der Anwendung und zum Informationsaustausch zwischen den Komponenten benötigt man Protokolle und Dienste, die im Kommunikationsmodell festgelegt werden.

- Funktionsmodell
  Im Funktionsmodell wird der Gesamtkomplex ´Management´ in Funktionsbereiche zerlegt und für diese Funktionsbereiche entsprechende Management-Funktionen definiert.

2.2.2.2 Architektur (Informations-, Organisations- und Kommunikationsmodell)

Obgleich der Vielfalt der Lösungen im Bereich Management, basieren alle Lösungen auf einer grundlegenden Architektur. Diese gemeinsame Architektur definiert vier wesentliche Komponenten eines Managementsystems: Management-Anwendung, Manager, Agent und Managed Object.

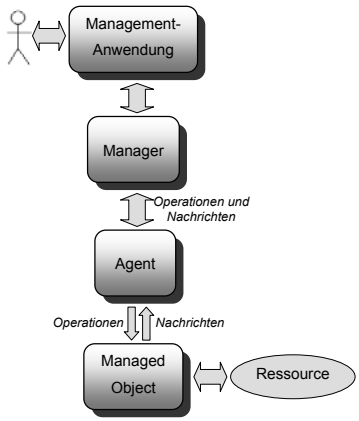

Abbildung 2-36 Management-Architektur

Die verwalteten logischen und physikalischen Ressourcen werden durch so genannte ´Managed Objects´ repräsentiert, welche eine Abstraktion auf die für das Management relevanten Aspekte der Komponente darstellen. Die Grenze (boundary) eines Managed Objects bestimmt, welche Details der Ressource einem Management-System zugänglich sind.

"A managed object is the abstracted view of a resource that represents its properties as seen by (and for the purpose of) management"[1].

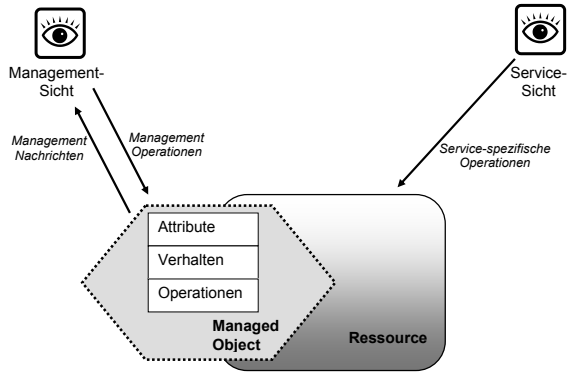

Abbildung 2-37 Beziehung zwischen Ressource und Managed Object

Demnach werden Managed Objects beschrieben durch:

- Attribute, in denen Managementinformationen gespeichert werden können, wie zum Beispiel Parameter oder Zustandsgrößen. Attribute beschreiben den Zustand eines Managed Objects. Sie können sich ändern, wenn sich der Zustand des realen Pendants ändert und gezielt durch Management-Operationen manipuliert werden.
- Operationen, die auf dem Managed Object und ggf. auf der überwachten Komponente selbst ausgeführt werden können, wie zum Beispiel ein Geräte-Reboot. Operationen ermöglichen den Zugriff auf ein Managed Object, so dass die Menge der möglichen Operationen die Leistungsfähigkeit und Komplexität des Managed Objects beeinflusst. Typische Operationen sind Lesen (get), Schreiben (set), Erzeugen (create) und Löschen (delete).

[1] Vgl. ISO 7498-4.

- Verhalten, mit dem das Managed Object auf die auszuführenden Operationen reagiert. Es bestimmt die Semantik sowie die Interaktion mit dem realen Pendant und wird in der Regel verbal definiert. Beim Verhalten wird zwischen internen und externen Stimuli unterschieden. Hinter dem Begriff ´Interne Stimuli´ verbergen sich Vorgänge, die direkt zwischen der Ressource und dem MO ablaufen. So kann zum Beispiel ein abgelaufener Zähler eine interne Stimulation auslösen. Externe Stimuli treten auf, wenn Managementoperationen von anderen Stationen auf das MO wirken, und dieses dann durch definierte Verhaltensweisen reagiert.
- Nachrichten, mit denen ein Managed Object über den Eintritt eines Ereignisses informieren kann, wie zum Beispiel Schwellwertüberschreitungen oder Medienmangel.

Die Gesamtheit der Managementinformationen, die in und durch die Managed Objects bereitgestellt wird, wird in der Regel als Management Information Base (MIB) bezeichnet.

"The set of managed objects within a system, together with their attributes, constitutes that´s management information base."[1].

Die Management-Anwendung stellt das Frontend zum menschlichen Part des Managements, also einem Administrator oder Operator, dar. Es handelt sich hierbei um eine mehr oder weniger komplexe grafische Benutzeroberfläche, von der aus das gesamte System überwacht, gesteuert und manipuliert werden kann. Der Manager nimmt Anfragen und Aufträge der Management-Anwendung entgegen und leitet diese an den Agenten weiter. Auf der anderen Seite erhält er vom Agenten sowohl Antworten auf diese Operationen als auch Meldungen, welche er an die Management-Anwendung weiterleitet. Der Agent verwaltet die Managed Objects und damit verbunden die MIB, führt die vom Manager initiierten Operationen aus und übermittelt die Rückgabewerte an diesen. Darüber hinaus ist es die Aufgabe des Agenten, Meldungen von den Managed Objects an den Manager weiterzuleiten.

Weitere Komponenten dieser Architektur, die nicht als Basiskomponenten betrachten werden, sind:

- Das Management-Protokoll, welches Agent und Manager zum Austausch der Management-Informationen benutzen.

[1] Vgl. ISO 7498-4.

- Die Ressourcen, welche durch Managed Objects abstrahiert werden.

Sowohl Anzahl als auch Relationen der Komponenten sowie deren Verteilung differieren in den verschiedenen Management-Architekturen. Auf diese Aspekte wird in den entsprechenden Unterkapiteln näher eingegangen.

2.2.2.3 Management-Funktionen (Funktionsmodell)

Um den weit gefassten Bereich des Managements zu untergliedern, haben sich im Zuge der Standardisierung der OSI fünf Teildisziplinen herauskristallisiert, die jedes Managementsystem erfüllen sollte, auf die in den folgenden Teilabschnitten eingegangen werden soll:

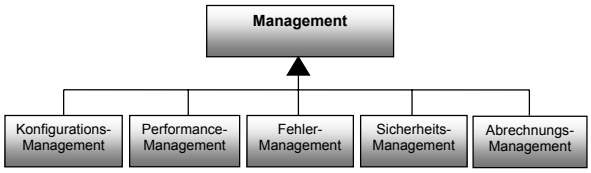

Abbildung 2-38 Funktionale Gliederung des Managements

- Konfigurations-Management
  Die Aufgaben des Konfigurationsmanagements liegen in der Bestandsaufnahme und Inventarisierung der zu überwachenden Ressourcen. Hierzu muss zunächst eine Datenbank (Management Information Base) erstellt werden, in der jede Ressource mit den zugehörigen Parametern gespeichert ist. Eine weitere Aufgabe des Konfigurationsmanagements liegt darin, die durch Performance-Management und Fehlermanagement entdeckten Schwachstellen durch entsprechende Umkonfiguration der Ressourcen zu beseitigen. Ebenso ist es, je nach Dynamik der zu überwachenden Ressourcen bzw. des Netzwerkes notwendig, die Parameter den Umständen entsprechend oft zu ändern.
- Performance-Management
  Diese Form des Managements dient zur Überwachung der Leistungsfähigkeit der Ressourcen. Performance-Probleme können entdeckt und durch das Konfigurationsmanagement behoben werden. So kann nicht nur Fehlern

vorgebeugt, sondern auch die Effizienz der Systeme erhöht werden, so dass mit vorhandenen Mitteln ein Maximum an Performance erzielt werden kann.

- Fehler-Management
  Mit zunehmender Größe und Komplexität eines Netzwerkes bzw. der enthaltenen Ressourcen wird auch die Fehlererkennung immer schwieriger und gleichzeitig bedeutender. Daher hat das Fehlermanagement die Aufgabe, einen Defekt möglichst schnell zu erkennen und zu lokalisieren, um diesen mit geeigneten Mitteln zu beheben.
- Sicherheits-Management
  Diese Form des Managements ist für den Schutz der Daten vor unberechtigten Zugriffen zuständig. Zum einen können durch unbeabsichtigte oder mutwillige Aktionen Modifikationen und Fehler im System hervorgerufen werden, zum anderen können durch Angriffe vertrauliche Informationen ausgespäht oder manipuliert werde. Sowohl Modifikationen und Fehler als auch Angriffe, gilt es so schnell wie möglich zu erkennen und entsprechende Gegenmaßnahmen einzuleiten. Durch geeignete Maßnahmen, wie Verschlüsselung und Authentifikation müssen diese Probleme minimiert werden.
- Abrechnungs-Management
  Die Aufgabe des Abrechnungsmanagements besteht darin, die Kosten für in Anspruch genommene Ressource zu ermitteln, so dass diese auf Benutzer oder Kostenstellen im Unternehmen gerecht verteilt werden können. Dazu muss jeder Zugriff auf das Netz bzw. die Ressourcen protokolliert und entsprechend ausgewertet werden.

2.2.3 OSI-Management / CMIP

Wie bereits angesprochen, begann die International Organisation for Standardization (ISO) 1977 mit der Festlegung einer prinzipiellen Architektur von Telekommunikationssystemen. Die Ergebnisse wurden 1984 als das allgemein bekannte ISO/OSI-Basisreferenzmodell[1] veröffentlicht und sollen als Gerüst für die Entwicklung von Standards in diesem Bereich dienen. Die dem Modell zugrunde liegende Idee ist die Aufteilung der komplexen Gesamtaufgabe in überschaubare Teilaufgaben. Dies führte zu einer hierarchischen Gliederung eines Telekommunikationssystems in sieben Schichten, welche gemeinsam die Kommunikation verschiedener Rechnersysteme über räumliche Distanzen hinweg ermöglichen sollen. Die folgende Abbildung soll, aufgrund des allgemeinen Bekanntheitsgrades und der Verbreitung dieses Modells, zur Erklärung genügen.

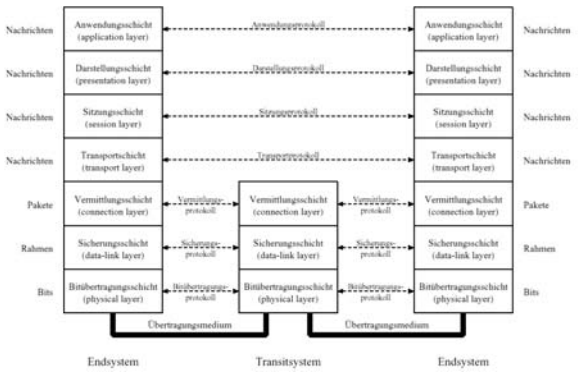

Abbildung 2-39 ISO/OSI-Basisreferenzmodell (Quelle: [ZWSG94],S.14)

1980 startete die ISO erste Aktivitäten im Bereich Management als Teil der Entwicklung dieses ISO/OSI-Referenzmodells. Die langwierigen Standardisierungs-Bemühungen beim OSI-Management führten letztendlich erst Ende der 80er Jahre

[1] Vgl. ISO 7489-1.

zum Erfolg, d.h. zur Herausgabe des ersten Standards zum OSI-Management. Diese wurden und werden seitdem noch verbessert und erweitert. Das OSI-Management ist die erste und bisher einzige Management-Architektur, welche alle vier Teilmodelle (siehe *Kapitel 2.2.2.1 – Teilmodelle einer Managementarchitektur*) detailliert spezifiziert. Damit kommt dem OSI-Management, wie auch dem ISO/OSI-Schichtenmodell im Bereich Rechnernetze, die Rolle einer Referenzarchitektur zu. Die praktische Relevanz des OSI-Managements ist, im Vergleich zum Internet-Management, insbesondere im Bereich privater Netze, eher gering (auch hier sei der Vergleich zum ISO/OSI-Schichtenmodell erlaubt). Bei öffentlichen Netzen dagegen ist es als Basis für das Telecommunications Management Network (TMN) von Bedeutung. Einen Überblick über die zum OSI-Management gehörenden Standards gibt folgende Abbildung.

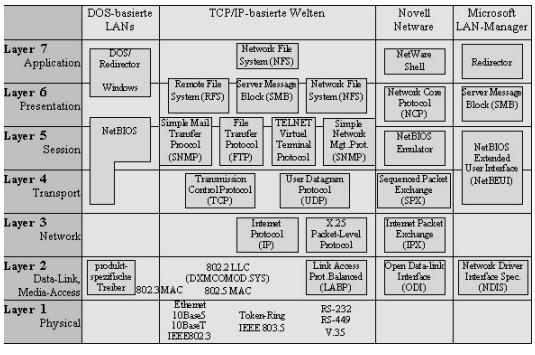

Abbildung 2-40 OSI-Management Standards

Unter allen Standards, die von der ISO entworfen wurden, sind die OSI Systems Management Standards die umfangreichsten und komplexesten. Diese Standards gliedern sich in fünf Kategorien:

- OSI Management Framework and Overview
  ISO 7489-4 gibt eine allgemeine Einführung in Managementkonzepte und ISO 10040 gibt einen Überblick über die restlichen Dokumente.

- CMIS/CMIP
  ISO 9505 beschreibt den Common Management Information Service (CMIS), der Dienste für Managementanwendungen bereitstellt. ISO 9506 beschreibt das zugehörige Protokoll, das Common Management Information Protocol (CMIP).
- Systems Management Functions
  ISO 10164-1/-14 definiert spezielle Funktionen, die vom OSI Systemmanagement ausgeführt werden können.
- Management Information Model
  ISO 10165-1/-6 beschreibt die Management Information Base (MIB), welche alle OSI-Objekte bezüglich des Netzwerkmanagements repräsentiert.
- Layer Management
  ISO 10733/10737 definieren Managementinformationen, Dienste und Funktionen bezogen auf bestimmte OSI-Schichten.

2.2.3.1 Organisationsmodell

Das Organisationsmodell des OSI-Managements stellt sich folgendermaßen dar.

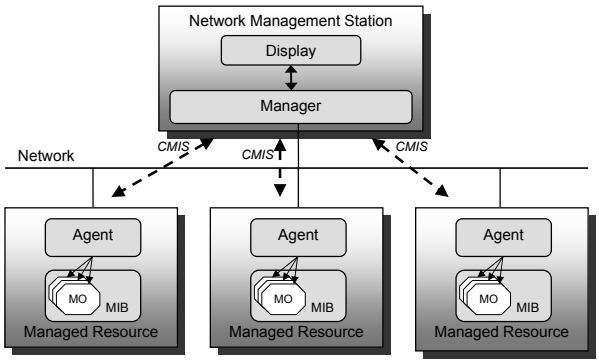

Abbildung 2-41 Organisationsmodell des OSI-Managements

Grundsätzlich existieren zwei verschiedene Ansätze der Kooperation zwischen den Komponenten einer verteilten Anwendung. Die erste Möglichkeit wird ´peer-to-peer´

genannt und bewältigt die zu lösende Aufgabe durch gleichberechtigte, symmetrische Kommunikation zwischen den Ressourcen. Die zweite Möglichkeit, das ´Manager-Agenten-Modell´, welches auch beim OSI-Management benutzt wird, ist eine asymmetrische Kommunikationsbeziehung, bei der die beiden Rollen Manager und Agent unterschieden werden (siehe hierzu auch *Kapitel 2.2.2.2 – Architektur (Informations-, Organisations- und Kommunikationsmodell)*). Das OSI-Management bietet zwei Möglichkeiten zur Verteilung der Management-Komponenten. Die erste Variante geht davon aus, dass sich sowohl MIB als auch der Agent ´auf´ der zu überwachenden Ressource befinden, d.h. soll zum Beispiel ein Automat überwacht werden, befinden sich diese Komponenten auf der den Automaten steuernden Einheit. Eine andere Möglichkeit bietet das Proxy-Konzept, bei dem sich MIB und Agent auf einem, der zu überwachenden Ressource vorgeschalteten Stellvertreter befinden. Der Manager befindet sich auf einer entfernten Station und kommuniziert mit Hilfe des CMIS (Common Management Information Service) mit einer gewissen Anzahl von Agenten.

OSI bietet die folgenden Möglichkeiten, Managed Objects (MOs) in Domänen zusammenzufassen:

- Organisationsdomäne
  Hier werden Managed Objects nach funktionalen Gesichtspunkten (z.B. MOs für Sicherheits- oder Fehlermanagement) oder innerhalb eines Funktionsbereiches zur Durchsetzung einer gemeinsamen Managementpolitik (Policy) zusammengefasst. Organisationsdomänen können sich überlappen, d.h. ein MO kann Mitglied mehrer Organisationsdomänen sein.
- Verwaltungsdomäne
  Im Gegensatz zu Organisationsdomänen, dürfen sich Verwaltungsdomänen nicht überschneiden. Zweck von Verwaltungsdomänen ist die Einrichtung und Manipulation von Organisationsdomänen sowie die Kontrolle des Steuerungsflusses zwischen Organisationsdomänen.

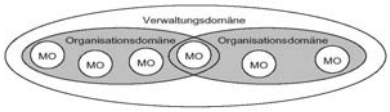

Abbildung 2-42 OSI-Domänenmodell

2.2.3.2 Informationsmodell

Auch im OSI-Modell werden die Repräsentanten der zu überwachenden Ressourcen als Managed Objects (MOs) bezeichnet. Ein MO kann ein Gerät (z.B. einen Router), Teile eines Gerätes (z.B. die einzelnen Ports eines Switches), nicht-physische Ressourcen (z.B. Einträge in Management-relevante Log-Files) oder auch einzelne Verbindungen eines verbindungsorientierten Protokolls repräsentieren. Die Schnittstelle vom MO zum umgebenden System, bzw. der das MO kontrollierenden Agenten wird Managed Object Boundary (MOB) genannt. Von Bedeutung für die Management-Architektur ist dabei nur die Information, die an der MOB sichtbar ist, die Implementierung der MOs hingegen ist nicht standardisiert. Die Gesamtheit aller sich in einem System befindlichen MOs wird auch hier als Management Information Base (MIB) bezeichnet. Der Rahmen zum Entwurf der MIB und zur Festlegung der Datentypen wird von der Structure of Management Information (SMI) in ISO 10165 (siehe auch *Abbildung 2-40 OSI-Management Standards*) gestellt, welche stark mit objektorientierten Konzepten arbeitet.

SMI legt folgende Punkte fest:

- Modell für Managed Objects und deren Attribute
  Hier erfolgt eine detaillierte Beschreibung der MOs, deren Attribute, Operationen, Meldungen und Verhaltensweisen. Die Spezifizierung der MOs geschieht durch Erstellung von Managed Object Classes (MOCs), aus denen dann die MOs instanziiert werden. Weiterhin sind auch hier folgende objektorientierte Ansätze vorhanden:
  - Kapselung: Managementanwendungen haben nur über das MO Zugang zur Ressource, um diese zu überwachen und zu verwalten.
  - Vererbung: Die Vererbung ermöglicht die Unterklassenbildung aus bereits vorhandenen Klassen. Weiterhin ist es damit möglich eine Hierarchie zu bilden, die die Ressourcen widerspiegelt. Die MOCs werden dann im so genannten ´Inheritance-Tree´ abgelegt.
  - Aggregation: Es ist möglich eine ´ist-enthalten-in´-Beziehung zwischen MOs zu definieren, wobei ein MO immer nur in genau einem anderen MO enthalten sein kann. Die MOCs werden dann im so genannten ´Containment-Tree´ abgelegt.

  - Allomorphismus: Dieser Ansatz bietet die Möglichkeit verschiedenartige Managed Objects zu implementieren, die aber dasselbe Verhalten gegenüber der Managementstation aufweisen. So können beispielsweise alte Ressourcenmodelle durch neue ersetzt werden, indem das Verhalten der alten Elemente weiterhin simuliert wird.
- Prinzipien der Bennennung der Managed Objects
  Jede Managed Object Class (MOC) ist durch einen eindeutigen Identifikator gekennzeichnet. Dieser Bezeichner ist eine Kette von Integerzahlen, die der jeweiligen Klasse einen eindeutigen Platz im so genannten ´ISO Registration-Tree´ zuordnet. Jede Definition einer MOC muss in diesem Baum registriert werden, um sie eindeutig identifizieren zu können. In diesem weltweit eindeutigen Baum sind die Definitionen für MOCs, Attributdefinitionen, Operationen, Meldungen usw. abgelegt. Diese Struktur weist jeder registrierten Definition einen ASN.1-Object Identifier[1] der Form $i_1.i_2. \ldots .i_n$ zu. Die einzelnen Ziffern entsprechen den Knotenmarkierungen im Registration-Tree. So ist zum Beispiel der Knoten des DoD[2] durch 1.3.6 zu erreichen. Dieser weltweit eindeutige Baum, der auch beim Internet-Management (siehe *Kapitel 2.2.4 – Internet-Management / SNMP*) eine wichtige Rolle spielt, stellt sich folgendermaßen dar.

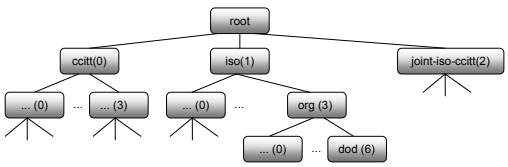

Abbildung 2-43 ISO Registration-Tree

- Konzepte von Managed Object Classes (MOC) und deren Beziehungen
  ISO 10165-1 schlägt Richtlinien für das Definitionsformat von MOs, Attributen, Meldungen usw. vor. Diese Richtlinien liegen in Form von Templates, die in der Beschreibungssprache OSI Abstract Syntax Notation One (ASN.1) definiert sind,

[1] ASN.1 ist eine formale Nachrichten-Beschreibungssprache für eine Reihe von Anwendungen. Weitere Informationen finden sich unter http://asn1.elibel.tm.fr.

[2] DoD ist das Department of Defense, das amerikanische Verteidigungsministerium.

vor. Ein Beispiel für die Definition einer MOC in ASN.1 soll durch die folgende Abbildung gegeben werden.

- Definition von SMI
  ISO 1065-2 definiert Basiselemente für die MIB Entwicklung. Zu diesen Elementen gehören Attribute, Objektklassen und Meldungstypen. Die Beschreibung erfolgt auch hier in der ASN.1.

```
Hub     MANAGED OBJECT CLASS
        DERIVED FROM ISO/IEC 10165-2:Top;
        CHARACTERIZED BY:
            BEHAVIOUR:          ……..
            ATTRIBUTES          HubID                   GET,
                                NumberOfRelays          GET,
                                RelayActive             GET,
                                TimeSinceHubSystemReset GET,
                                HubResetTimeStamp       GET,
                                HubHealth               GET,
                                GroupMap                GET;
            ACTIONS             ResetHubSystemAction,
                                RelayChangeoverAction;
            NOTIFICATIONS       HubHealth,
                                GroupRelayConfigChange,
                                ProprietaryExtensionAlarm;
        REGISTERED AS
        iso(1)std(0)iso8802(8802)csma(3)hubmgmt(18)objectclass(0)hubobjectclass(X)
```

Abbildung 2-44 Beispiel für die MOC Definition

2.2.3.3 Kommunikationsmodell

Die wichtigste Funktionalität im OSI-Management ist der Austausch von Managementinformationen zwischen zwei Instanzen mit Hilfe eines Protokolls. Diese Funktionalität spiegelt sich im Common Management Information Service Element (CMISE) wieder, welches wiederum andere Dienstelemente zur Ausführung benötigt. Die Abwicklung des Informationsaustausches zwischen den Anwendungsprozessen wird in Schicht sieben des ISO/OSI-Basisreferenzmodells (siehe *Abbildung 2-39 ISO/OSI-Basisreferenzmodell*) durchgeführt. Die folgende Abbildung zeigt die Einbettung der Dienstelemente in die Anwendungsschicht.

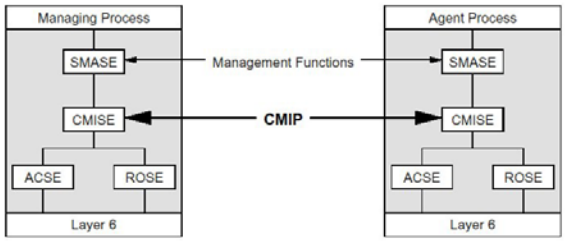

Abbildung 2-45 OSI-Dienstelemente in Schicht 7 (Quelle: [STAL93],S. 214)

Das Systems Management Application Service Element (SMASE) stellt eine Vielzahl von Managementfunktionen bereit und bildet zugleich die Schnittstelle zu den Anwendungsprozessen. Es stützt sich auf CMISE und das zugehörige Common Management Information Protocol (CMIP), die den Zugriff und die Manipulation von Managed Objects ermöglichen. CMISE benötigt für die eigene Funktionalität zwei weitere Dienstelemente: das Association Control Service Element (ACSE) zur Verbindungsverwaltung, und das Remote Operations Service Element (ROSE) zur Durchführung von Managementoperationen auf entfernten Systemen.

Durch CMISE werden Dienste definiert, die für das OSI-Management durch jede Komponente vorgesehen sind. Diese sind durch Dienstprimitive aufrufbar, die in drei Gruppen gegliedert sind:

- Management Operation Services
  Diese Services dienen der Ausführung von Management-Operationen. In dieser Dienstklasse werden die folgenden Dienstprimitive zur Verfügung gestellt:
  - M-GET Lesen von Attributwerten
  - M-SET Modifikation von Attributwerten
  - M-CANCEL-GET Abbruch eines M-GET Dienstaufrufes
  - M-ACTION Aufruf einer Operation eines Managed Objects
  - M-CREATE Erzeugung eines Managed Objects
  - M-DELETE Löschen eines Managed Objects
- Management Notification Services
  Aufgabe dieser Dienstklasse ist es, in einem Netzwerk über Ereignisse zwischen

zwei Partnerinstanzen zu berichten. Diese Instanzen fungieren hierbei in der Manager/Agenten-Beziehung. Als einziges Dienstprimitiv findet sich:

- M-EVENT-REPORT Ereignisweiterleitung vom Agenten an Manager

- Management Association Services
  Die Dienste dieser Klasse kontrollieren die Anwendungsassoziationen zwischen den beteiligten Systemen. In erster Linie sind dies Assoziationsaufbau und -abbau. Beteiligte Dienste sind:
  - M-INITIALIZE richtet eine Assoziation mit einem Partner ein
  - M-TERMINATE beendet eine Assoziation mit einem Partner normal
  - M-ABORT bricht die Assoziation abrupt ab

2.2.3.4 Funktionsmodell

Managementaufgaben sind vielseitiger Art und können daher Funktionsbereichen, so genannten System Management Functional Areas (SMFAs) zugeordnet werden. ISO schlägt eine Einteilung in fünf Kategorien, den so genannten FCAPS vor. FCAPS steht hierbei für:

- Fault-Management
- Configuration-Management
- Accounting-Management (Abrechnungs-Management)
- Performance-Management
- Security-Management

Dies sind die bereits in *Kapitel 2.2.2.3 – Management-Funktionen (Funktionsmodell)* beschriebenen Managementfunktionen, die mehr oder weniger ausgeprägt, in jeder Technologie wieder zu finden sind. Die Zuordnung zu den entsprechenden ISO-Referenzen stellt sich folgendermaßen dar:

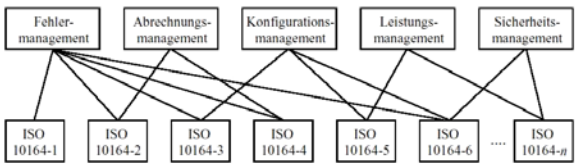

Abbildung 2-46 OSI Funktionsbereiche (FCAPS)

2.2.4 Internet-Management / SNMP

Als 1983 die TCP/IP-Protokollfamilie vom Amerikanischen Verteidigungsministerium (DoD) zum Standard erklärt wurde, war dies die Geburtsstunde des Internets. Die folgende Abbildung zeigt das TCP/IP-Modell (inklusive der wichtigsten Standards) im Kontext des ISO/OSI-Modells. Auch für dieses weit verbreitete und allgemein bekannte Modell soll diese Abbildung als Erklärung genügen.

ISO/OSI Schichten								DoD Schichten
Anwendung	Hypertext Transfer	File Transfer	Electronic Mail	Terminal Emulation	Domain Name Service	Network Management	Network File System	Prozess Applikation
Darstellung	Hypertext Transfer Protocol	File Transfer Protocol	Simple Mail Transfer Protocol	Telnet Protocol	Domain Name System	Simple Network Management Protocol	XDR RFC (1014)	
Sitzung	(HTTP) RFC 1945	(FTP) RFC 59	(SMTP) RFC 821	(Telnet) RFC 854	(DNS) RFC 1034	(SNMP) 1157	RPC RFC (1057)	
Transport	Transmission Control Protocol (TCP) RFC 793					User Datagram Protocol (UDP) RFC 768		Host to Host
Netzwerk / Verbindung	Internet Protocol (IP) RFC 791		Address Resolution Protocol (ARP) RFC 826		Reverse Address Resolution Protocol (RARP) RFC 903	Internet Control Message Protocol (ICMP) RFC 792		Internet
Verbindungssicherung	Ethernet, Token Ring, DQDB (802.X), FDDI							lokales Netzwerk oder Netzzugriff
Bit-Übertragung	Übertragungsmedium							

Abbildung 2-47 TCP/IP-Modell

Mit dem Begriff SNMP (Simple Network Management Protocol) wird eine Menge von Spezifikationen im Bereich Netzwerkmanagement bezeichnet, sowohl das Protokoll selbst als auch die Definition der Datenbasis und die mit SNMP verbundenen Konzepte. Wie man in *Abbildung 2-47 TCP/IP-Modell* sieht, ist SNMP auf Applikationsebene in diesem Modell enthalten, d.h. es ist Bestandteil der TCP/IP-Protokollsuite. An dieser Stelle sollen, wie auch beim OSI-Management, nur die grundlegenden Konzepte beschrieben werden, da eine detaillierte Beschreibung dieser Technologie den Rahmen dieser Arbeit sprengen würde. Die Beschreibung beschränkt sich weiterhin auf die erste Version von SNMP, da diese trotz aller Bemühungen seitens der Standardisierungsgremien, noch die am weitesten verbreitete Version ist.

2.2.4.1 Organisationsmodell

Das Organisationsmodell gleicht im Bereich Architekturmodell dem des OSI-Managements. Auch hier existieren Manager, Agenten, Managed Objects und MIB, welche auf die gleiche Art und Weise auf den Ressourcen verteilt werden. Nur im Bereich der Domänenbildung finden sich im Internet-Management bislang noch keine Ansätze.

2.2.4.2 Informationsmodell

Während das ISO-Informationsmodell auf einem konsequent objektorientierten Ansatz beruht und damit zwar komplex aber mächtig ist, werden die Managed Objects im Internet-Management als einfache Variablen beschrieben, die in einem Registrierungsbaum zusammengefasst sind. Es können also in der MIB nur einfache Datentypen, d.h. Skalare und Felder aus Skalaren, gespeichert werden. Es ist nicht erlaubt, komplexe Datenstrukturen zu kreieren bzw. abzufragen, da diese aufgrund einfacherer Implementierbarkeit und Kompatibilität vermieden werden. Alle Managed Objects in der MIB werden auch hier mit Hilfe von ASN.1 definiert. Da allerdings die Einfachheit gefordert wurde, wird nur eine Teilmenge dieser Sprache benutzt. Die Struktur eines Managed Objects wird, im Gegensatz zur Definition einer Managed Object Class (MOC) im ISO-Modell, über die Knoten im ISO-Registrierungsbaum nachgebildet, wodurch sie beliebig kompliziert werden kann. Dies stellt einen gravierenden Nachteil von SNMP dar.

Managed Objects im Internet-Management werden immer unterhalb des Knotens 1.3.6.1 im ISO-Registrationsbaumes registriert. Das nachfolgende Beispiel zeigt die Definition einer einfachen SNMP-Variablen und deren Position innerhalb des Baumes.

OBJECT	*sysDescr { system 1 }*
SYNTAX	*Octet String*
DEFINITION	*This value should include the full name, version ID, software operating-system and networking software.*
ACCESS	*read-only*
STATUS	*mandatory*

Abbildung 2-48 Beispiel für die Definition einer SNMP-Variable

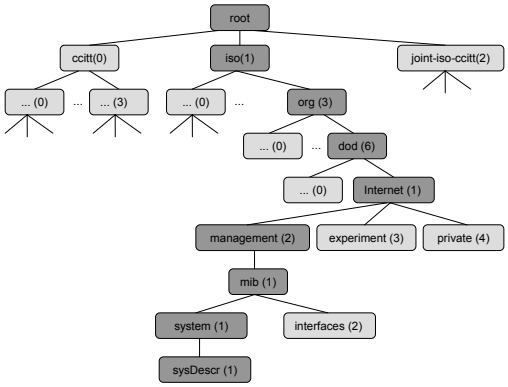

Abbildung 2-49 Registrierung der definierten SNMP-Variable

2.2.4.3 Kommunikationsmodell

Managementinformationen werden beim Internet-Management über das Simple Network Management Protocol (SNMP) ausgetauscht. Es ist Teil der TCP/IP Protokollklasse und setzt im Allgemeinen auf dem User Datagram Protocol (UDP) auf. Da UDP ein verbindungsloser Dienst ist, ist die Verbindung zwischen Manager und Agent ebenfalls verbindungslos, wodurch jeder einzelne Nachrichtenaustausch eine eigene Transaktion darstellt. Die folgende Abbildung verdeutlicht die typische Protokollkonfiguration.

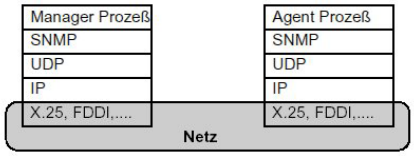

Abbildung 2-50 Einordnung SNMP in das TCP/IP-Modell

SNMP definiert nur fünf Typen von Meldungen, welche zwischen Manager und Agenten ausgetauscht werden können:

- GET-REQUEST
  Die Get-Operation dient zum Lesen von einer oder mehreren MIB-Instanzen, d.h. ein oder mehrere Variablenwerte werden abgerufen.
- GET-NEXT-REQUEST
  Die Get-Next-Operation erlaubt das Lesen des Wertes der nächsten MIB-Instanz gemäß der lexikographischen Ordnung. Mit Hilfe aufeinander folgender Get-Next-Operationen ist es möglich, eine komplette MIB ohne Kenntnis der Struktur zu durchlaufen.
- SET-REQUEST
  Die atomare Set-Operation schreibt Werte in eine oder mehrere MIB-Instanzen. Mit Hilfe dieser Operation können auch neue MIB-Instanzen erzeugt werden.
- GET-RESPONSE
  Hierbei handelt es sich um die vom Agenten zum Manager zurückgegebene Antworten auf die get-, get-next- und set- Operationen.
- TRAP
  Mit der Trap-Operation kann ein Agent ein Ereignis anzeigen, d.h. dieses wird vom Agent an den Manager übermittelt. Der Empfang einer Trap-Nachricht wird nicht bestätigt, und Trap-Nachrichten können während der Übertragung verloren gehen.

Damit ergibt sich ein differenzierteres Kommunikationsmodell, welches in der folgenden Abbildung dargestellt wird.

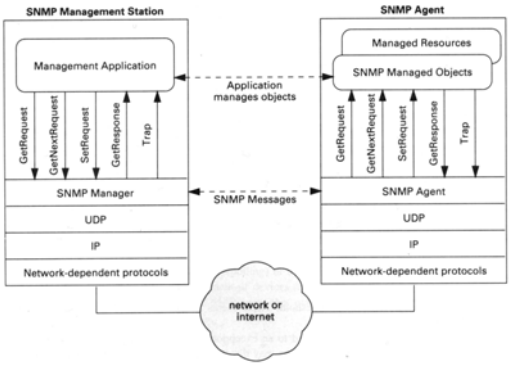

Abbildung 2-51 SNMP Kommunikationsmodell (Quelle: [STAL93], S.75)

2.2.4.4 Funktionsmodell

Das Funktionsmodell gehört zu den Teilmodellen einer Managementarchitektur, die vom IAB[1], im extremen Gegensatz zum OSI-Ansatz, bislang stark vernachlässigt wurde. Natürlich ist es, auch wenn nicht definiert, möglich, die fünf FCAPS-Funktionsbereiche mit SNMP abzudecken.

[1] Das Internet Activities Board (heute: Internet Architecture Board) ist eine technische Beratungsgruppe der Internet Society. Weitere Informationen unter http://www.iab.org.

2.2.5 Vergleich OSI- und Internet-Management

Dieses Kapitel liefert eine Gegenüberstellung der Managementansätze der ISO (International Standardization Organisation) und des IAB (Internet Activities Board) als den heute bedeutendsten Organisationen, die sich um eine herstellerübergreifende Management-Architektur bemühen. Der wesentliche Unterschied besteht in der Art und Weise, wie ISO und IAB das Ziel eines herstellerübergreifenden Managements erreichen wollten. Vor diesem Hintergrund müssen die aufgezeigten Unterschiede bewertet werden. Die ISO, die als erste Standardisierungsorganisation mit den Arbeiten zu einem offenen Netzwerkmanagement begonnen hat,

- verfolgt ein flexibles und mächtiges Lösungskonzept und
- unterstützt einen möglichst universellen Ansatz.

Das heißt, die in den verschiedenen Teilmodellen der Architektur getroffenen Festlegungen sollen zur Lösung beliebig komplexer Managementaufgaben genutzt werden können. Im Gegensatz zur Universalität des ISO-Ansatzes steht die Pragmatik des IAB-Ansatzes. Die Ziele waren hierbei:

- Minimierung der Komplexität und damit der Entwicklungskosten
- die Basis für eine schnelle Lösung für die Managementproblematik im Internet
- eine spätere Migration zum OSI-Management

Anfangs wurde die einfache Implementierbarkeit des Ansatzes über sämtliche anderen Anforderungen gestellt, wodurch die heute bekannten Lücken und konzeptionellen Schwächen des SNMP bedingt sind. Andererseits trugen der Einsatz und die daraus resultierenden praktischen Erfahrungen von SNMP-basierten Werkzeugen dazu bei, dass heute an dieser Stelle Konzepte des OSI-Managements als angemessen erachtet und umgesetzt werden.

Die folgende Tabelle soll die wichtigsten Unterschiede zwischen beiden Ansätzen zusammenfassend darstellen.

	OSI-Management (CMIP)	Internet-Management (SNMP)
Protokoll	CMIP	SNMP
Verbreitung	geringe Verbreitung, außer im Telekommunikationsbereich mit TMN	sehr weit verbreitet
Komplexität	hoch	gering
Effizienz	gut	akzeptabel
Implementierung	schwierig	einfach
Darunter liegender Dienst	ROSE, verbindungsorientiert, gesichert, dadurch größerer Netzwerk-Overhead	UDP, verbindungslos, ungesichert
Informationsansatz	objektorientiert	variablenorientiert
Kommunikations-Objekte	Objekte	Skalare
Zustandsinformationen	werden vom Agent verwaltet	werden vom Manager verwaltet
Ereignisbestätigung	bestätigt oder unbestätigt	unbestätigt
Kommandoausführung	möglich	nur über Umwege möglich
Sicherheit	durch darunter liegende Dienste gegeben	keine
ASN.1-Unterstützung	volle Unterstützung	Teilmengen-Unterstützung
Selektion von MOs	Dynamisch durch Scoping und Filtering	nur direkt über Namen oder Nachfolger

Tabelle 2-5 Unterschiede OSI- und Internet-Management

2.2.6 JMX – Die Management-Architektur von Morgen?

Einer der am schnellsten wachsenden Bereiche innerhalb des IT-Betriebs ist das Management verteilter IT-Infrastrukturen. Die Gründe hierfür liegen insbesondere in der manuell kaum mehr beherrschbaren Komplexität der Netze, Systeme und Anwendungen. Als Beispiel sei hier die Konfiguration und Administration von Customer Relationship Systemen (CRMS) oder Supply Chain Management Systemen (SCMS) genannt. Solche schlecht beherrschbaren Systeme, die zu kritischen Elementen in der Wertschöpfungskette eines Unternehmens oder gar in der mehrerer Unternehmen gemeinsam, geworden sind, verursachen im Fehlerfall hohe Kosten und enormen Aufwand. Es ist daher heute wichtiger denn je, durch geeignete Management-Systeme die Überwachung und Steuerung der gesamten IT-Infrastruktur auf allen Ebenen zu gewährleisten (siehe *Kapitel 2.2.1.1 – Der Begriff ´Management´*).

Die klassischen Managementarchitekturen, welche in den vorherigen Teilkapiteln beschrieben wurden, sind diesen neuen Anforderungen kaum gewachsen. SNMP ist zwar sehr weit verbreitet, hat aber in vielen Bereichen (siehe *Kapitel 2.2.5 – Vergleich OSI- und Internet-Management*) massive Defizite. Allein der variablenorientierte Infomationsansatz, mit allen darauf basierenden Unzulänglichkeiten, führt dazu, dass in vielen Bereichen, zum Beispiel im Bereich des Managements von Bankperipheriegeräten (siehe *Kapitel 3 – Ein Überwachungssystem für Bankperipheriegeräte*), proprietäre Management-Protokolle entwickelt werden, um auf spezifische Bedürfnisse besser eingehen zu können. Der OSI-Ansatz ist zwar in vielen Bereichen, insbesondere durch den objektorientierten Ansatz, besser geeignet, aber durch die geringe Verbreitung und die hohe Komplexität mit den damit verbundenen Kosten auch keine Alternative.

Beiden klassischen Ansätzen ist gemeinsam, dass sie als zentralisierte, plattformbasierte Architekturen realisiert sind, d.h., dass von einer zentralen Managementplattform (Manager) aus eine große Anzahl von Agenten verwaltet wird. Diese Agenten besitzen aber nur einfache Zugriffsschnittstellen und sind gewöhnlich nicht in der Lage, eine Vorverarbeitung der Managementdaten, die sie sammeln, durchzuführen. Dieser Ansatz wird als statisch bezeichnet, da das Vorhandensein vordefinierter Managementfunktionen vorausgesetzt wird, und eine Änderung dieser Funktionalität zur Laufzeit nicht möglich ist. So ist bei einer Änderung der Managementanforderungen oder der Managementinformationen in der MIB eine

Neukompilierung des Agenten oder die Installation eines neuen Agenten notwendig. Aus dem zentralisierten Managementansatz ergeben sich folgende Nachteile:

- Die zentrale Managementplattform stellt einen ´single point of failure´ dar.
- Beim Verlust der Verbindung zwischen Manager und Agent geht jede Möglichkeit der Steuerung und Überwachung oder sogar die Funktionalität der überwachten Ressource verloren.
- Durch die große Anzahl an Agenten und die fehlende Verdichtung der Managementdaten durch diese kommt es zum Teil zu erheblicher Netzlast.
- Der zentrale Manager wird u.a. aus diesem Grund häufig überlastet.
- Die Anzahl der Applikationen, die auf einer zentralen Managementplattform zur Ausführung gebracht werden können, ist begrenzt.
- Auf neue Managementanforderungen kann nur schwerfällig reagiert werden.
- Die Architektur ist unflexibel.

Diese Schwachpunkte können durch den so genannten ´Management by Delegation´-Ansatz behoben werden. Dieser dezentrale Ansatz geht davon aus, dass die zu überwachenden Systeme immer leistungsfähiger werden, wodurch die Möglichkeit entsteht, immer mehr Überwachungs- und Kontrollaufgaben dezentral durchführen zu können. Weiterhin soll die Funktionalität dynamisch auf die Agenten verlagert werden können, um schnell und effizient auf Änderungen der Management-Anforderungen reagieren zu können.

Heutige Netze und insbesondere die auf diesen Netzen operierenden, verteilten Anwendungen unterliegen einem immer rascheren Wandel, der sich auch im Bedarf an ein flexibles, skalierbares, dynamisches, offenes, standardisiertes und eben den neuen Anforderungen angepasstes Management äußert. Hier kommen Technologien wie CORBA, WBEM und JMX ins Spiel. Siehe hierzu auch *Kapitel 2.2.1.5 – Standards und Entwicklungen.*

Die Java Management Extensions (JMX) ist ein von Sun Microsystems und Partnern wie IBM, TIBCO, Bea, Computer Associates und Borland entwickeltes Management-Framework der ´dritten Generation´. Es basiert auf frühen Sun-Entwicklungen im Management-Bereich, wie der bereits 1995, also vor der Freigabe der Sprache Java, entwickelten Java Management API (JMAPI). JMX liegt zurzeit in Version 1.0 als

optionales Paket für J2SE[1] vor und es existieren neben der Referenzimplementierung bereits zwei kommerzielle, auf JMX basierende Produkte: Das Java Dynamic Management Kit 4.2 (JDMK) von Sun und Tivoli-JMX von IBM.

2.2.6.1 Organisationsmodell

Das JMX-Organisationsmodell baut auf dem allgemeinen, von der OSI definiertem, Organisationsmodell auf, d.h. auch hier existieren die vier grundlegenden Komponenten auf drei Ebenen.

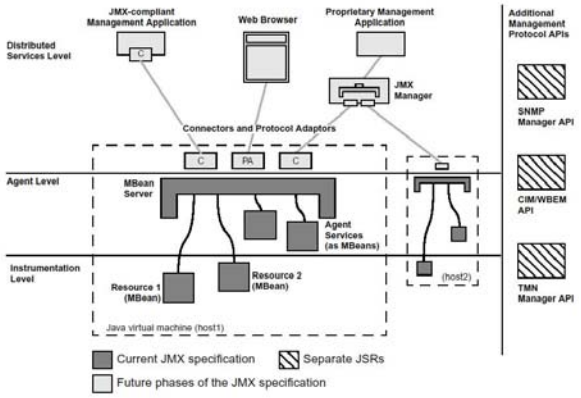

Abbildung 2-52 JMX-Architektur (Quelle: [JMX2], S.21)

Die drei Ebenen der JMX-Architektur und die Kernkomponenten, auf die in den Folgeabschnitten näher eingegangen werden soll, stellen sich folgendermaßen dar:

- Instrumentation Level (Managed Objects Level)

  Die Spezifikation dieser Ebene legt die Anforderungen an den Repräsentanten einer Ressource, d.h. an ein Managed Object fest, die erfüllt werden müssen,

[1] J2SE steht für Java 2 Standard Edition. Weitere Informationen unter http://java.sun.com/j2se.

damit diese über JMX verwaltet werden kann. Managed Objects (MOs) werden in der JMX Terminologie als Managed Beans (MBeans) bezeichnet.

- Agent Level

  Auf dieser Ebene werden JMX Agenten spezifiziert, deren Aufgabe es ist, MBeans zu verwalten, zu steuern und dem Manager zugänglich zu machen. Ein JMX Agent besteht aus: MBean Server, Agent Services, Konnektoren und Protokoll-Adapter

- Distributed Services Level (Manager Level)

  Auf dieser Ebene werden sowohl JMX Manager als auch die Schnittstellen zwischen dem JMX Agenten und den Managementanwendungen bzw. dem vorgeschalteten Manager definiert. Diese Schnittstellen werden durch Konnektoren und Protokoll-Adapter realisiert. Diese Schicht ist in der aktuellen Spezifikation noch nicht enthalten, befindet sich aber in der Entwicklung.

Zusätzlich zu den drei genannten Ebenen der JMX Architektur sieht die JMX Spezifikation APIs vor, die als Schnittstelle zu anderen Managementarchitekturen fungieren sollen. Im Gegensatz zu den Protokolladaptern, welche es ermöglichen mit Managementanwendungen, die Protokolle wie zum Beispiel SNMP benutzen, auf die JMX Infrastruktur zuzugreifen, ermöglichen es diese ´Additional Management Protocol´ APIs, Ressourcen einer anderer Managementarchitektur in ein JMX-basiertes Managementsystem einzubinden. Solche APIs sind u.a. für SNMP und CIM/WBEM vorgesehen. Die Spezifikation dieser APIs erfolgt aber außerhalb der JMX Spezifikation in eigenen Java Specification Requests (JSRs) des Java Community Process (JCP).

Im Bereich der Domänenbildung bietet JMX die Möglichkeit MBeans, wie auch im OSI-Modell, in einfache Domänen zusammenzufassen, die den OSI-Verwaltungsdomänen entsprechen. (siehe *Kapitel 2.2.3.1 - Organisationsmodell*)

2.2.6.2 Informationsmodell

Wie auch das OSI-Modell basiert JMX auf einem konsequent objektorientierten Ansatz, d.h. Konzepte wie Kapselung, Vererbung, Aggregation und die Definition von Assoziationen auf MBean-Basis sind uneingeschränkt möglich. MBeans sind auch hier Repräsentanten einer zu überwachenden Ressource, welche eine Anwendung, ein Dienst, ein Gerät usw. sein kann. Die Definition der MBeans erfolgt aber nicht,

wie beim OSI- und SNMP-Management, mit Hilfe von ASN.1, sondern basiert auf dem einfachen JavaBeans-Konzept[1]. Ein Managed Bean (MBean) ist ein Java-Objekt, welches eine spezielle Managementschnittstelle implementiert und bestimmten Entwurfsmustern entspricht. Laut Spezifikation ist ein MBean folgendermaßen definiert:

- Es muss eine konkrete und öffentliche Klasse sein.
- Es muss einen öffentlichen (public) Konstruktor besitzen.
- Es muss der JavaBeans Spezifikation genügen.
- Es muss ein korrespondierendes MBean Interface bzw. das DynamicMBean Interface implementieren.
- Es kann das NotificationBroadcaster Interface implementieren, wenn es Ereignisse an andere Komponenten melden soll.

Dadurch wird sichergestellt, dass der JMX Agent in der Lage ist, das MBean zu instanziieren und dessen Managementschnittstelle zu erkennen, die alle Attribute, Operationen, Benachrichtigungen und Konstruktoren enthält, um das MBean bzw. die zugrunde liegende Ressource zu verwalten. Ein MBean, welches der Spezifikation genügt, kann von jedem JMX Agenten verwaltet werden. Es existieren weiterhin keinerlei Restriktionen, wo sich die zu ladende MBean Klasse befindet, d.h. sie kann sich innerhalb des Klassenpfades der virtuellen Maschine des Agenten oder auf einem entfernten Rechner befinden. So ist es möglich MBeans dynamisch über das Netzwerk zu laden, um die Funktionalität des Agenten zur Laufzeit zur erweitern oder neuen Managementanforderungen anzupassen.

JMX definiert vier MBean Typen:

- Standard MBean
  Diese MBean Art ist am einfachsten zu entwerfen und zu implementieren. Die Management-Schnittstelle wird bei diesem MBean Typ zur Entwurfszeit festgelegt und ist daher im späteren Betrieb statisch.
- Dynamic MBean
  Für permanenten Änderungen unterworfene Managementanforderungen sind

[1] Das JavaBeans Konzept ist das Java Komponentenmodell. Weitere Informationen finden sich unter http://java.sun.com/products/javabeans.

Standard MBeans aufgrund ihrer statischen Management-Schnittstelle nicht geeignet. Dynamic MBeans offenbaren ihre Management-Schnittstelle zur Laufzeit, indem diese dann durch fest definierte Zugriffsfunktionen bekannt gegeben wird.

- Open MBean
  Open MBeans sind Dynamic MBeans mit einem beschränktem, fest definiertem Satz an Datentypen und detaillierten Meta-Informationen. Das Ziel hierbei ist, sich selbst beschreibende MBeans zu erhalten, die sehr einfach benutzt werden können. Dieser MBean Typ ist in der JMX 1.0 Spezifikation noch nicht vollständig spezifiziert.
- Model MBean
  Diese MBean Art, welche auch vom Dynamic MBean abgeleitet ist, stellt ein generisches und konfigurierbares Management-Template für Managementobjekte dar, um den Implementierungsaufwand zu reduzieren. Zum Zeitpunkt der Erzeugung können die Signaturen durch XML[1], OMG IDL[2] oder Java beschrieben werden. Weiterhin bietet das Model MBean die Möglichkeit der persistenten Speicherung.

[1] Diese Metasprache wird ausführlich in *Kapitel 2.1.2 – XML (Extensible Markup Language)* vorgestellt.

[2] Die Interface Definition Language (IDL) ist eine Schnittstellenbeschreibungssprache aus der CORBA-Welt.

Zur Verdeutlichung soll auch hier ein Beispiel für die Definition eines Managed Objects, am Beispiel eines Standard MBeans, gezeigt werden.

```
public interface PrinterMBean {
        public boolean  getOnline();
        public void     setOnline(boolean status);
        public int      getSheetsInTray();
        public void     printStatusPage();
}

public Printer implements PrinterMBean {
        // attributes
        private boolean online;         // READ / WRITE
        private int     sheetsInTray;   // READ ONLY

        // constructors
        public Printer() { this.online  = false; this.sheetsInTray = 0; }

        // Accessor-Methods
        public boolean  getOnline()                 { return this.online; }
        public void     setOnline(boolean status)   { this.online = status; }
        public int      getSheetsInTray()           { return this.sheetsInTray; }

        // Methods
        public void printStatusPage() {
                … <access real hardware printer ressource> …
                this.sheetsInTray--;
        }
}
```

Abbildung 2-53 Beispiel einer Standard MBean Implementierung

Für Nachrichten, mit denen ein MBean über den Eintritt eines Ereignisses informiert, definiert die JMX Spezifikation ein Notification Modell, welches sich auf das allgemein bekannte Java Event Modell stützt. Ereignisse (Events) sind Java Objekte, die von einer Ereignisquelle (Event Source) an eine oder mehrere Ereignissenken (Event Listener), die sich vorher hierfür registriert haben müssen, verschickt werden. Events, die in der JMX Terminologie als Notifications bezeichnet werden, können sowohl von MBeans als auch vom MBean Server an beliebige Ereignissenken versendet werden. Der MBean Server hat die Möglichkeit eintreffende Ereignisse mit einem so genannten Notification Filter zu filtern.

Die Verwaltung der Gesamtheit der MBeans, also der MIB, wird, wie bereits angesprochen, durch den MBean Server übernommen. Diese MIB stellt er dem Manager zur Verfügung, der nun entsprechend auf den MBeans operieren und Notifications von diesen empfangen kann. Der MBean Server stellt aber immer nur die Management-Schnittstelle des MBeans zur Verfügung, niemals die direkte Objektreferenz. Bei der Registrierung eines MBean beim MBean Server, bekommt

dieses einen eindeutigen identifizierenden Objektnamen zugewiesen. Dieser wird später benutzt, um das Objekt zu identifizieren, auf dem Managementoperationen ausgeführt werden sollen oder von dem Notifications eintreffen. Der Objektname besteht aus dem Domänennamen, welcher das MBean einer entsprechenden Domäne zuordnet und einer nichtleeren Menge von Name/Wert-Paaren, welche zum Beispiel den Namen des MBeans, sowie weitere Parameter enthalten können.

Die Selektion von MBeans geschieht bequem über Objektnamen-Filter oder Suchausdrücke, wobei jeweils Wildcards benutzt werden können. So könnte ein Objektnamen-Filter alle MBeans selektieren, bei denen der Objektname ´MyMBeans´ enthält und ein entsprechender Suchausdruck könnte alle MBeans selektieren, bei denen das Attribut ´online´ den Wert ´false´ hat.

Wie auch beim OSI-Modell, ist es möglich, Relationen zwischen MBeans zu definieren. Dies ist vorteilhaft, wenn unterschiedliche MBeans interagieren sollen, oder wenn bestimmte Funktionen eines MBeans von der Existenz eines anderen abhängig sind.

Bei Agent Services, welche außer dem MBean Server weitere wichtige Bestandteile des Agent Levels sind, handelt es sich um Objekte, die den MBean Server um Managementfunktionalitäten erweitern. Sie stellen Dienste zur Verfügung, welche von MBeans oder vom MBean Server zur Verrichtung ihrer Aufgaben genutzt werden können. Es ist empfehlenswert, Agent Services selbst als MBeans zu implementieren, um diese auch einheitlich administrieren und überwachen zu können. Agent Services können sowohl durch die JMX Spezifikation definiert als auch durch Drittanbieter vertrieben werden. Elementare Agent Services, welche in jeder JMX Implementierung zwingend enthalten sein müssen, sind:

- Dynamic Class Loading Service
  Mit dem M-Let Service (Management Applet) bietet JMX die Möglichkeit, MBeans dynamisch über das Netzwerk zu laden und im MBean Server zu registrieren. Dies ermöglicht es die Funktionalität des Agenten zur Laufzeit zur erweitern oder neuen Managementanforderungen anzupassen.
- Relation Service
  Der Relation Service stellt die Funktionalität bereit, welche notwendig ist, um Relationen zwischen MBeans zu definieren, zu pflegen und aufzulösen. Weiterhin

überwacht dieser Dienst die Einhaltung der definierten Relationen und informiert registrierte Ereignissenken über Änderungen an Relationen.

- Monitoring Service
  Dieser Dienst ermöglicht es, MBean Attribute zu überwachen und bei Veränderungen der Attribute registrierte Ereignissenken darüber zu informieren.
- Timer Service
  In vielen Fällen ist es notwendig, Aktionen in regelmäßigen Abständen oder zu wohldefinierten Zeitpunkten auszuführen. Zu diesem Zweck können sich MBeans beim Timer Service registrieren, um sich zum gewünschten Zeitpunkt benachrichtigen zu lassen.

Die nachfolgende Abbildung zeigt einen detaillierten Einblick in den Agent-Level.

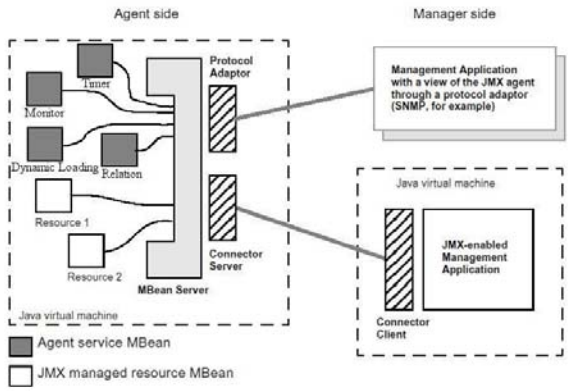

Abbildung 2-54 Aufbau des JMX Agenten (Quelle: [JMX2], S.102)

2.2.6.3 Kommunikationsmodell

Manager interagieren mit dem Agenten über so genannte Konnektoren und Protokoll-Adapter, welche als MBeans offen gelegt sind. Aufgrund dieser Komponenten benötigen JMX Agenten keinerlei Kenntnisse über die Managementanwendung, die auf sie zugreift. Es muss nur der entsprechende Konnektor oder Adapter im MBean Server registriert sein. Konnektoren ermöglichen die Kommunikation eines JMX Managers mit einem JMX Agenten in der Art, dass die MBean Interfaces unverändert an den Manager weitergereicht werden. Adapter setzen Anfragen von Seiten einer Nicht-JMX-konformen Managementanwendung (z.B. SNMP-Manager, Webbrowser) in JMX-Anfragen um. In der Gegenrichtung müssen natürlich auch Antworten und Notifications von JMX in das korrespondierende Protokoll umgesetzt werden. Bei einem Protokoll-Adapter handelt es sich also um einen Protokoll-Wandler zwischen JMX und einem beliebigen anderen Protokoll. Die Definition der Konnektoren und Adapter ist nicht Gegenstand der JMX Spezifikation Version 1.0, wird aber in der nächsten Version zu finden sein.

2.2.6.4 Funktionsmodell

Das Funktionsmodell ist in den JMX Spezifikationen nicht explizit definiert. Natürlich ist es trotzdem möglich, die fünf FCAPS-Funktionsbereiche (siehe *Kapitel 2.2.3.4 – Funktionsmodell*) mit JMX abzudecken.

2.2.6.5 Zusammenfassung

Zusammenfassend sollen hier noch einmal die Vorteile der JMX Architektur vorgestellt werden:

- Es ergeben sich durch die Java-Plattform bedingte Vorteile wie Plattformunabhängigkeit, Einfachheit, Robustheit, typsichere Referenzen, automatische Speicher-Bereinigung usw.
- Ressourcen werden ohne großen Aufwand leicht überwachbar.
- Es handelt sich um eine dynamisch erweiterbare Management-Architektur, da sowohl Managed Objects als auch Agenten-Dienste, dynamisch geladen und integriert werden können.

- JMX Agenten sind dynamisch skalierbar, d.h. sie können überall dort eingesetzt werden, wo eine JVM[1] installiert ist, d.h. vom Handy bis zum High-End Server. Die Größe des Agenten hängt nur von den notwendigen Diensten und den Repräsentanten der zu überwachenden Ressourcen ab.
- Es besteht die Möglichkeit einer einfachen Integration existierender Management-Lösungen (bereits jetzt sind JMX Agenten über HTML Browser sowie SNMP- und WBEM-Manager überwachbar).
- Es existiert die Möglichkeit des Web-based Managements, da einfache Web-Interfaces mit Hilfe des HTML-Adapters sehr einfach und dynamisch erzeugt werden können.
- Es ist möglich Dienste zu realisieren, mit deren Hilfe der Agent Aufgaben des Managers übernimmt und damit eine Vorverarbeitung und Verdichtung der Daten durchführt.
- MBeans sind vollständig zu JavaBeans kompatibel, so dass sie innerhalb jeder JavaBean Entwicklungsumgebung verwendet werden können.

In Version 1.0 der JMX-Spezifikation, in der insbesondere im Distributed Services Level (Manager Level) noch nicht alle Komponenten spezifiziert sind, sind je nach Verwendung eine Reihe Schwächen enthalten:

- JMX bietet keinen standardisierte Lokalisierung- oder Namensdienste an. Eine Möglichkeit die hier von Sun propagiert wird, ist die Benutzung von Jini[2] in Kombination mit JMX. Dieses spontane Auffinden von Ressourcen und Netzwerkdiensten mit Jini, welche dann mit JMX überwacht werden, wird als ´Sun Spontaneous Management´ bezeichnet.
- Da JMX kein eigenes Informationsmodell definiert, sind im Bereich der Adapter oft noch Anpassung von Hand vorzunehmen.
- JMX Agenten sind nicht in der Lage in einem Netz zu migrieren, d.h. sie werden auf einem System instanziiert und befinden sich dort bis zu ihrer Zerstörung. JMX ist daher als Plattform für mobile Agenten ungeeignet.

[1] Die Java Virtual Machine (JVM) ist integraler Bestandteil der Java-Plattform und dient der Ausführung von Java-Bytecode.

[2] Diese Technologie und Dienstinfrastruktur ermöglicht das spontane Auffinden und Interagieren von Ressourcen in einem Netz. Weitere Informationen unter http://www.sun.com/jini.

- Es fehlt ein einheitliches Sicherheitskonzept. Es werden lediglich für bestimmte Adapter einfache Sicherheitsmechanismen zur Verfügung gestellt.

Der mit JMX konkurrierende Ansatz ist CIM/WBEM[1] der DMTF[2], welcher im Gegensatz zu JMX auf XML setzt. Die Zeit wird zeigen, welcher und ob sich einer dieser Standards endgültig durchsetzten kann. JMX ist wie gesagt auf jedem System, welches über eine JVM verfügt, einsetzbar und profitiert daher von der breiten Ausrichtung der Java Plattform, welche mittlerweile von Embedded Systems bis hin zu High-End Servern reicht. Somit ist es möglich, dass JMX in der Zukunft sowohl zur Steuerung von Haushaltsgeräten als auch bei der Administration von unternehmensweiten Anwendungen eingesetzt wird. JMX wird seine Hauptrolle wahrscheinlich im Management von Java-Anwendungen und hier insbesondere im Management von J2EE Applikationsservern spielen. Sun sieht vor, JMX in die nächste J2EE Version als Managementplattform für die J2EE-Plattform zu integrieren. Dies spiegelt sich im JSR 77 – J2EE Management Specification wieder.

2.2.7 Integrierte Managementsysteme

Im Bereich der Managementsysteme existieren eine Vielzahl von Lösungen für die verschiedensten Bereiche und Branchen. So gibt es spezielle Managementsysteme für das Anwendungsmanagement, andere für das Netzwerkmanagement und wieder andere für den Bereich des Informationsmanagements. Der Trend geht seit einiger Zeit aber eindeutig in Richtung integrierte Managementsysteme, die alle Bereiche des Managements (siehe *Kapitel 2.2.1.1 – Der Begriff ´Management´*) abdecken. Weiterhin sollen bereits bestehende Systeme und zugekaufte Bausteine problemlos in diese Lösung integriert werden können und eine einheitliche Benutzeroberfläche existieren. Ein System, welches diese Eigenschaften besitzt, wird als integriertes Managementsystem bezeichnet und hat die in der folgenden Abbildung dargestellte grundlegende Architektur.

[1] Vgl. *Kapitel 2.2.1.5 – Standards und Entwicklungen.*

[2] Vgl. *Kapitel 2.2.1.5 – Standards und Entwicklungen*

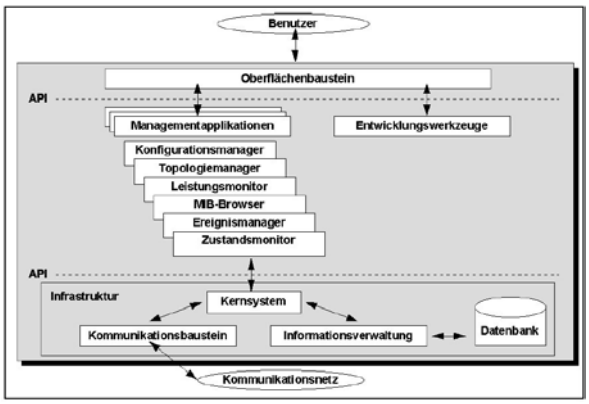

Abbildung 2-55 Integrierte Managementarchitektur (Quelle:[HHAS93],S.301)

Die Infrastruktur stellt verschiedene Basisdienste bereit, welche von allen Managementanwendungen benötigt werden. Das Kernsystem sorgt hierbei für die Kommunikation und Koordination der anderen Bausteine. Es ist unter anderem dafür zuständig, dass Anfragen an das jeweilige Modul weitergeleitet werden, für das sie bestimmt sind. Der Kommunikationsbaustein ermöglicht die Kommunikation mit Fremdsystemen wie Netzkomponenten, Endsystemen, Managementsoftware oder anderen Managementsystemen. Die Informationsverwaltung implementiert das (meist objektorientierte) Informationsmodell der Plattform und stellt Dienste für das Kreieren und Verwalten von Managementinformationen bereit. Zur Speicherung der Informationen bedient sich die Informationsverwaltung einer Datenbank, welche unter anderem die MIB enthält. Der Oberflächenbaustein ermöglicht dem Benutzer des Systems die Kommunikation mit dem System, d.h. den Zugang zu den umfangreichen enthaltenen Managementanwendungen. Darüber hinaus stellen integrierte Managementsysteme dem Benutzer einige Standardanwendungen sowie Entwicklungswerkzeuge zur Erstellung eigener individueller Applikationen zur Verfügung. Im Idealfall muss der Entwickler nur die richtigen Funktionen aus dem Infrastrukturbaustein mit den gewünschten Funktionen aus dem Oberflächenbaustein

verknüpfen und die benötigte Logik hinzufügen, um eine individuelle Management-Lösung zu erstellen, welche problemlos in das Gesamtsystem integriert werden kann. Weiterführende Literatur zum Bereich der integrierten Managementsysteme bietet [HHAS93].

Es existiert eine Reihe von mehr oder weniger geeigneten kommerziellen Standardsoftware-Paketen im Bereich der integrierten Managementsysteme. Die Key-Player in diesem Sektor sind:

- Hewlett Packard OpenView
- Bull OpenMaster
- IBM/Tivoli TME
- Computer Associate Unicenter TNG
- Sun SunNet Manager
- Cabletron Spectrum

2.3 Technologien zum Zugriff auf Bankperipheriegeräte / J/XFS

Tagtäglich benutzen hunderte von Menschen Bankautomaten: Karte einführen, Geheimnummer eingeben, Betrag wählen und schon wird das Geld ausgegeben. Danach noch schnell einen Kontoauszug holen. Für Kunden ist es ein einfacher Bestandteil des täglichen Lebens, für Banken und Automatenhersteller ist dies nicht ganz so einfach. Sowohl Entwicklung als auch die Integration von Bankperipheriegeräten wie Geldautomaten, Kontoauszugdrucker und Informationsterminals ist eine komplexe Angelegenheit. Banken tendieren dazu, diese Geräte und entsprechende Software bei einer Reihe von Herstellern zu erwerben. Oft haben oder hatten diese Hersteller unterschiedliche Schnittstellen zum Zugriff auf diese Geräte und verwendeten proprietäre Protokolle, wodurch die Entwicklung von Anwendungen ein detailliertes Know-how über gerätespezifische Interna erforderte. Seit einiger Zeit existieren verschiedene Bemühungen den Zugriff und die Nutzung von Bankperipheriegeräten zu standardisieren und damit u.a. die Komplexität zu reduzieren.

2.3.1 Historie

Selbstbedienungssoftware-Architekturen, welche die Technologien zum Zugriff auf Bankperipheriegeräte beinhalten, unterlagen im Laufe der Zeit einer Reihe von Veränderungen und entwickelten sich von monolithischen zu verteilten Anwendungen. Folgende Entwicklungen, die nicht jeweils das Ende der vorhergehenden Ära bedeuten, sondern koexistieren, lassen sich nachvollziehen:

- ca. 1980 – 1994 Cluster Architektur
- 1994 – 1998 Client/Server Architektur (WOSA/XFS)
- seit 1998 Network-Computing- und Thin-Client Architektur (J/XFS)

Bei den Cluster Architekturen beherrschten unflexible, teure und proprietäre Lösungen vor, so dass es, wie bereits angesprochen, immer wieder Standardisierungsbemühungen in diesem Bereich gab. Erste Erfolge konnte 1993 WOSA/XFS verbuchen. WOSA/XFS steht für Windows Open Services Architecture Extensions for Financial Services und ist eine von Microsoft, Siemens Nixdorf, NCR und DEC entwickelte Standardschnittstelle zum Zugriff auf Bankperipheriegeräte. Diese Schnittstelle ist Bestandteil der Windows Open Services Architecture (WOSA), bei der es sich um eine von Microsoft und Partnern definierte offene Client/Server Architektur handelt, welche im wesentlichen eine Beschreibung von

Anwendungsdiensten und Programmierschnittstellen beinhaltet. Mit diesem Ansatz wird die Anwendungsentwicklung durch Definition einheitlicher, herstellerneutraler Anwendungsschnittstellen (APIs) von anbieter- oder produktspezifischen Implementierungen entkoppelt. Andererseits werden den Herstellern klar definierte Service Provider Interfaces (SPI) angeboten, an die sie ihre spezifischen Produkte und Dienste entsprechend adaptieren können. Die im Rahmen von WOSA definierten Dienste betreffen heute vorrangig die Basisdienste für Datenhaltung, -zugriff und -integration (WOSA/ODBC), Messaging (WOSA/MAPI), sowie den Kommunikationsbereich (WOSA/SNA). Darüber hinaus werden in spezialisierten Gremien WOSA Extensions für vertikale Märkte definiert. Dies sind u.a. die Extensions for Real Time Market Data (WOSA/XRT) und eben WOSA/XFS. Die folgende Abbildung verdeutlicht diese Architektur.

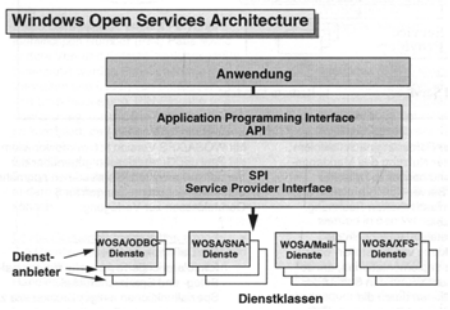

Abbildung 2-56 Windows Open Services Architecture (Quelle: [SNIWOSA])

Mit WOSA/XFS wurde ein Architekturrahmen geschaffen, der speziell auf die Belange der Finanzdienstleister ausgerichtet ist. Diese über den Gerätetreibern angeordnete Schicht, ist speziell für den Zugriff auf Selbstbedienungsgeräte im Bankenbereich konzipiert und unterstützt u.a. Geräte wie Drucker, Kartenlesegeräte und Geldausgabeautomaten. Ein ähnlicher Standard existiert mit JavaPOS (Java Point of Sales) für den Retail-Bereich, also zum Beispiel den Zugriff auf Kassensysteme.

Aber auch die Client/Server Architekturen bringen in diesem Bereich viele Nachteile mit sich:

- Softwareverteilung
- Softwarepflege
- kostspielige Administration der Clients
- Notwendigkeit der Anwendungsportierung aufgrund heterogener Umfelder
- hohe TCO (Total Cost of Ownership) aufgrund der o.a. Probleme

Um die angeführten Probleme zu reduzieren geht der Trend in Richtung Rezentralisierung durch Thin-Client Architekturen. Bei Thin-Clients, welche auch als ´Network Computer´ (NC) bezeichnet werden, handelt es sich um Clients mit minimaler Hardware-Ausstattung bezüglich Speicher, Festplatte und Prozessor. NCs besitzen die Eigenschaften eines einfachen Terminals mit grafischer Benutzeroberfläche. Das zentrale Softwaremanagement wird vom Server durchgeführt, die Business-Logik liegt auf dem Server und wird dort ausgeführt, so dass auf dem Client nur noch die Präsentation innerhalb eines Browsers liegt. Dieses Idealmodell kann in der Praxis nur in bestimmten Bereichen (E-Learning, Infoterminals, ´gemeinsames Surfen´ usw.) sinnvoll angewendet werden. Bei der Integration von Bankperipheriegeräte kann es leider nicht bestehen. So ist es Clients, welche nur mit einem Browser ausgestattet sind zum Beispiel nicht möglich auf die Hardware eines Geldautomaten zuzugreifen. Weiterhin ist der vorteilhafte dezentrale ´Management by Delegation´-Ansatz, bei dem Managementfunktionalität dynamisch auf die zu überwachende Ressource geladen wird, in einem solchen Szenario nicht realisierbar (siehe *Kapitel 2.2.6 – JMX - Die Managementarchitektur von Morgen?*). Deshalb erweitert man diese Architektur und teilt die Thin-Clients in zwei Formen ein: ´Ultra-Thin-Clients´, welche dem ursprünglichem Konzept der reinen Präsentations-Clients entsprechen, und so genannte ´Smart Clients´, welche zusätzlich zur Präsentation ein dynamisches Laden und Ausführen weiterer Funktionalität ermöglichen. Im Bereich der Integration von Bankperipheriegeräten sind aus den o.a. Gründen Smart-Clients sinnvoller.

Um diese Architekturanforderungen umzusetzen gründeten 1998 die Hauptanbieter von Lösungen in diesem Bereich wie Sun, Siemens Nixdorf (heute: Wincor Nixdorf), IBM, NCR und DeLaRue das so genannte ´J/XFS Forum´ (Extensions for Financial Services for the Java Platform). Ziel war es, eine Middleware für Banking-Applikationen zu erstellen und eine API zu definieren, welche von allen

teilnehmenden Unternehmen akzeptiert wird. Man entschied sich bei der Umsetzung schon zu Beginn aus folgenden Gründen für die Java-Plattform:

- wirkliche Plattformunabhängigkeit
- allgemeine Vorteile der Sprache Java wie Objektorientierung, Einfachheit, Robustheit, typsichere Referenzen, automatische Speicherbereinigung usw.
- sichere und stabile Applikationen
- dynamisch erweiterbare Applikationen (Applet-Technologie)
- niedrige Softwareentwicklungskosten

Insbesondere durch die Applet-Technologie war es nun möglich, Anwendungen oder Anwendungsteile dynamisch über das Netzwerk zu laden, wodurch zum Beispiel eine transparente Softwareverteilung und –update möglich wurde (software-on-demand). Weiterhin kommt es zu enormen Kosteneinsparungen, dadurch dass insbesondere die Softwareadministration und die Installation der Software nicht mehr direkt vor Ort, sondern zentral auf dem Server erledigt werden kann. Die berechneten Einsparungen beim Einsatz von Thin-Client-Architekturen gegenüber Client/Server-Architekturen liegen bei 25%. Diese Einsparungen kommen sowohl durch verminderte Administrations- und Verteilungskosten als auch durch Reduktion der Hardwarekosten zustande. Weiterhin beschloss man einen einheitlichen Ansatz für Bank- und Einzelhandelperipheriegeräte, so dass J/XFS auf WOSA/XFS und JavaPOS aufbauen sollte. Weiterhin sollte, durch Wrapping, eine Koexistenz der neuen mit den alten Lösungen möglich sein, um bereits getätigte Investitionen zu schützen.

Mitte des Jahres 1999 waren sowohl Systemdesign als auch Spezifikationen fertig. Das J/XFS Forum beschloss diese als offene Standards zu veröffentlichen[1] und man übergab diese dem CEN (Comité Européen de Normalisation)[2], wo dieser Ende des Jahres verabschiedet wurde. Daraufhin wurde der CEN J/XFS Workshop gegründet, welcher seitdem die Weiterentwicklung und Verbreitung dieses Standards verfolgt.

[1] Weitere Informationen unter www.jxfs.com.

[2] Das CEN ist das Europäische Komitee für Normung in Brüssel. Weitere Informationen unter www.cenorm.be.

2.3.2 Technologie, Architektur und Funktionsweise

Eine Reihe von Zitaten soll die Bedeutung der J/XFS Technologie sowohl für Banken als auch Hersteller, verdeutlichen:

„J/XFS is the platform independent Java language standard that will be used to communicate with and controls self-service devices."[1]

"With J/XFS, developers need to know whether they are writing for a printer or cash dispenser, but they don´t need to know who manufactured the device or how their application should talk to it, all is hidden in the architecture."[2]

"The obvious benefit for financial institutions is that developers can focus on the business aspects of the application without having to worry about the idiosyncrasies of particular peripheral devices."[3]

"J/XFS should give banks much greater flexibility when buying peripherals. They can change suppliers and install a new device, without having to make changes to the applications that run on it. […] they are free to pick the best type of device for their purposes."[4]

"The Mission is to produce a Banking Industry standard for a banking financial I/O device subsystem that supports 100% pure Java applications while leveraging existing standards."[5]

"The product contains anything needed to write a Java banking application which needs to access banking peripherals (it does not include, of course, any device drivers for specific hardware). The device services for the end devices are written and supplied by each vendor who sells J/XFS compliant hardware."[6]

[1] Vgl. [JXCO], S.3.

[2] Vgl. [OFM01], S.1.

[3] Vgl. [OFM01], S.2.

[4] Vgl. [OFM01], S.2.

[5] Vgl. [JXCO], S.3.

[6] Vgl. [JXSO], S.7.

Die bereits angesprochene Hersteller- und Plattformunabhängigkeit wird durch folgendes Schaubild verdeutlicht.

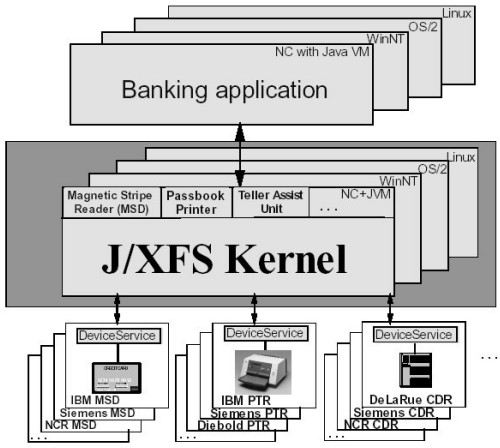

Abbildung 2-57 J/XFS Unabhängigkeiten (Quelle: [JXSO], S.7)

Die bisher von J/XFS unterstützten Gerätetypen sind:

- verschiedene Drucker (Kontoauszug, Quittung, Sparbuch, usw.)
- verschiedene Scanner
- Geldautomat
- Infoterminal
- Tastenfelder
- Kartenlesegeräte (Chip- und Magnet-Karten)
- Textein- und Ausgabegeräte
- Kameras, Sensoren, Alarmanlagen und Sonderelektronik

2.3.2.1 Architektur

Die J/XFS Architektur stellt sich folgendermaßen dar.

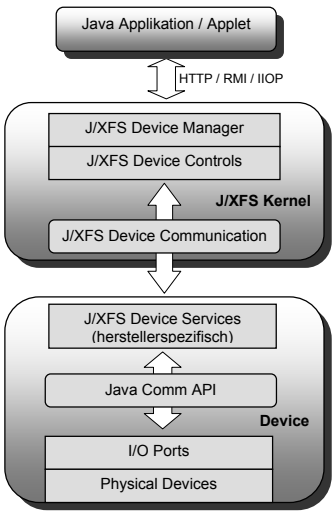

Abbildung 2-58 J/XFS Architektur

Über wohldefinierte Schnittstellen kann eine Java Applikation bzw. ein zuvor vom Server geladenes Applet, auf den J/XFS Kernel zugreifen. Dieser besteht aus Device Manager, Device Controls und der transparenten Kommunikationsschicht. Der Kernel benutzt diese Kommunikationsschicht, um auf den gerätetyp-spezifischen Device Service zuzugreifen, welcher vom Gerätehersteller oder Service Provider zur Verfügung gestellt wird. Dieser ist wiederum in der Lage, die eingehenden Kommandos in hersteller- und gerätespezifische Kommandos umzuwandeln und diese über eine entsprechende API, wie die Java Communication API, an die I/O Ports (COM, LPT usw.) der Hardware zu senden. Natürlich ist auch der umgekehrte Weg, also vom Gerät zur Applikation vorgesehen, um bestimmte Ereignisse (z.B.

Füllstandsunterschreitungen) zu melden. Grundsätzlich existieren zwei Arten von Applikationen, welche J/XFS nutzen: zum einen diejenigen, welche direkt auf dem Gerät ausgeführt werden (z.B. Benutzerschnittstelle eines Bankautomaten) und zum anderen diejenigen, welche nicht lokal sondern zur Überwachung der Geräte eingesetzt werden. Die Komponenten der J/XFS Architektur sollen in den folgenden Unterabschnitten detailliert erläutert werden:

- J/XFS Device Manager
  Diese Komponente ist die zentrale Kontrollinstanz des J/XFS Frameworks, durch die jede Kommunikation zwischen Gerät und Applikation geführt wird. Die Hauptaufgaben des Device Managers sind:
  - Verwaltung einer Liste von Geräten, Diensten und Kommunikationen
  - transparente Kontrolle der Geräte, Kommunikationsbeziehungen und Dienste, d.h. zum Beispiel Organisation der (möglicherweise gleichzeitigen) Zugriffe auf ein Gerät
  - Verwaltung einer zentralen Konfigurationsdatenbank
- J/XFS Device Control
  Diese oberste Schicht der Gerätehierarchie dient dem Zugriff der Applikation auf die Geräteschnittstelle, d.h. die Applikation nutzt die Device Controls, um mit dem physikalischem Gerät zu kommunizieren. Es findet also eine Entkopplung der Applikation von der physikalischen Ressource statt. Diese Schnittstelle gliedert sich in Gerätetypen, welche über spezifische Eigenschaften, Operationen und auszulösende Ereignisse verfügen. Es bietet folgende Funktionalität:
  - Öffnen und Schließen einer Verbindung zum Gerät
  - Durchreichen von Ereignissen bzw. Nachrichten an die Applikation
  - Durchreichen der Kommandos an den Device Service
  - exklusive Nutzung eines Gerätes
  - einfache Utility-Funktionen

  Aufgrund der Kompatibilität mit dem Java Komponentenmodell JavaBeans, kann ein Device Control als JavaBean dargestellt werden. (Ein Vergleich mit den MBeans der Managementarchitekturen bietet sich an. Siehe hierzu auch *Kapitel 2.2.2 Terminologie und grundlegende Konzepte – Management-Architektur*.)

- J/XFS Device Communication
  Diese austauschbare Schicht bietet die Möglichkeit der transparenten Kommunikation zwischen Device Control und Device Service. Sie versteckt die Komplexität der Netzwerkkommunikation und ist u.a. für die Steuerung gleichzeitiger Zugriffe auf Device Services zuständig.
- J/XFS Device Service
  Diese Schicht entkoppelt die Device Control Schicht von den Spezifika der physikalischen Peripheriegeräte. Der hersteller- und gerätespezifische Device Service bietet dem Device Control eine einheitliche Schnittstelle und erlaubt durch Umsetzung in gerätespezifische Kommandos den direkten Zugriff auf die Hardware des Peripheriegerätes. Es handelt sich letztendlich also um einen spezialisierten Gerätetreiber. Zu jedem Gerätetyp ist in der ausgelieferten J/XFS Implementierung ein Device Service Dummy enthalten, welcher in eingeschränktem Maße für die Entwicklung benutzt werden kann.
- Weitere wichtige Bestandteile der J/XFS Architektur sind der J/XFS Logger und das J/XFS Repository. Der Logger dient dem Schreiben von Log-Dateien, welche die Kommunikation, die durchgeführten Aufrufe und eventuelle Fehler und Probleme protokollieren. Das Repository dient dem statischen und dynamischen Speichern von Konfigurationsdaten der J/XFS Umgebung. Es handelt sich hierbei zurzeit lediglich um eine strukturierte ASCII-Datei, die sowohl Administrator als auch Umgebung schreiben und lesen können.

Durch diese Architektur kann jede Java-Applikation transparent mit jeder J/XFS Implementierung und jedem J/XFS Device Service zusammenarbeiten. Dies bedeutet erstens, dass jede Java-Applikation auf jedes J/XFS-konforme Gerät zugreifen kann und das Gerät versteht sie. Zweitens versteht jeder in Java geschriebene Device Service die an ihn gerichteten API Aufrufe. Dies wird in der J/XFS Terminologie als ´compliance´ bezeichnet, wobei der erstere Fall als ´application compliance´ und der zweite als ´device compliance´ bezeichnet wird.

2.3.2.2 Funktionsweise

Der Zugriff auf ein Gerät via J/XFS soll anhand des folgenden Szenarios verdeutlicht werden.

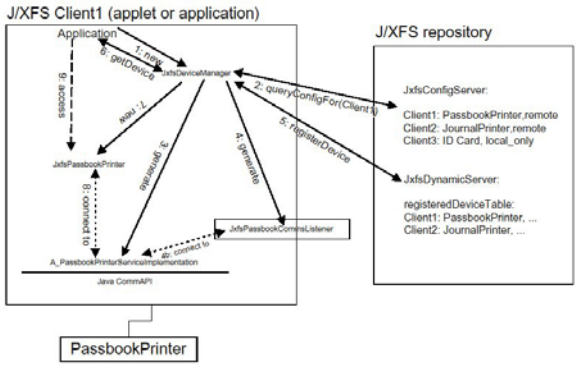

Abbildung 2-59 J/XFS Zugriff (Quelle: [JXBA], S.7)

Auf jedem J/XFS-Client wird zunächst durch die Applikation bzw. durch das Applet ein Device Manager Objekt generiert (1). Dieses initialisiert das J/XFS Subsystem durch Abfrage des J/XFS Repository nach den entsprechenden Konfigurationsdaten (2). Danach instanziiert es alle Device Services der lokal angeschlossenen Geräte (3). Sind Geräte für einen remote Zugriff konfiguriert, werden die entsprechenden Verbindungsobjekte generiert (4) und an ihren Device Service angeschlossen (4b). Sobald dies alles erfolgreich geschehen ist, werden die nun vorhandenen Device Services im Repository registriert, so dass andere J/XFS Clients diese nutzen können (5). Wenn die lokale Applikation bzw. das Applet jetzt Zugriff auf ein Gerät haben wollen, müssen sie den Device Manager nach diesem fragen. Dies geschieht mittels der getDevice()-Methode (6). Während dieser Methode generiert der Device Manager ein Device Control (7), findet den entsprechenden Device Service und bindet diesen an das Device Control (8). Das Device Control wird dann der Applikation übergeben, welche dieses nun zum Zugriff auf das Gerät benutzen kann (9).

Nach Durchlaufen des o.a. Szenarios ist es möglich, Operationen des Gerätes aufzurufen bzw. Ereignisnachrichten vom Gerät zu empfangen. Innerhalb von J/XFS gibt es zwei Arten von Methoden: synchrone und asynchrone. Erstere werden nur für einfache Funktionen benutzt, welche keinen direkten physikalischen Zugriff auf das Gerät benötigen, wie zum Beispiel Abfragen der Version eines Objektes oder Abfrage des Gerätestatus (dieser wird im Device Service gehalten). Jede Operation mit physikalischem Zugriff ist asynchron, d.h. dass zunächst nur eine Operations-ID zurückgegeben wird. Die Operation selbst wird in eine Warteschlange eingeordnet und in der Reihenfolge des Auftretens ausgeführt. Das Ergebnis der Operation ist in einem vom Device Service an die Applikation gesandten Ereignis enthalten, für die sich die Applikation zunächst registrieren muss. Das folgende Schaubild verdeutlicht den Ablauf einer J/XFS Operation.

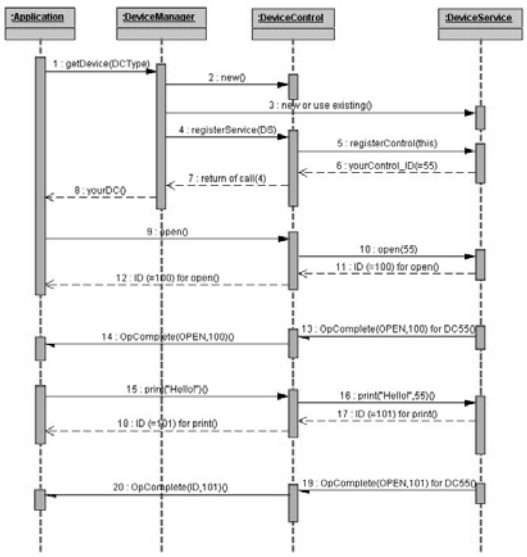

Abbildung 2-60 Ablauf einer J/XFS Operation

Der Vollständigkeit halber soll anhand eines kurzen Quelltextausschnittes die Funktionsweise und die Einfachheit des J/XFS Ansatzes demonstriert werden. Im folgenden Quelltext wird zunächst der Device Manager initialisiert und daraufhin von diesem das Device Control des gewünschten Gerätes, in diesem Fall ein Sparbuchdrucker, angefordert. Das Device Control bzw. die Verbindung zum Device Service muss zunächst geöffnet werden. Daraufhin wird mittels ´claim´ ein exklusiver Zugriff auf den Drucker gefordert und danach die gewünschten Operationen ausgeführt. Nach Beendigung der Operationen wird das Gerät wieder freigegeben, geschlossen, beim Device Manager deregistriert und letztendlich der Device Manager heruntergefahren.

```
myDeviceManager.initialize();
        myPassbookPrinter = myDeviceManager.getDevice("passbook");
                myPassbookPrinter.open();
                        myPassbookPrinter.claim();
                                myPassbookPrinter.printForm();
                                myPassbookPrinter.ctrlMedia();
                        myPassbookPrinter.release();
                myPassbookPrinter.close();
        myDeviceManager.deregisterDevice(myPassbookPrinter);
myDeviceManager.shutdown();
```

Abbildung 2-61 J/XFS Benutzungsbeispiel

2.3.3 Zusammenfassung

An dieser Stelle sollen noch einmal die wichtigsten Vorteile der J/XFS Technologie dargestellt werden:

- Vorteile durch die Sprache Java
  Die Sprache Java bietet eine Reihe von Vorteilen gegenüber der bei WOSA/XFS benutzten Sprache C. Sie ist einfach, darin geschriebene Applikationen sind robuster und es existiert eine automatische Speicherbereinigung.
- Reduzierte Kosten für Client-Konfiguration und Client-Administration
  J/XFS ermöglicht Thin-Client-Architekturen, welche eine serverseitige Konfiguration und Administration ermöglichen. (siehe *Kapitel 2.3.1 – Historie*)
- Plattformunabhängigkeit
  Die einzige Anforderung an J/XFS Applikation besteht in einer Java Virtual Machine. Diese JVM kann sowohl Bestandteil eines Browsers oder des Betriebssystems sein als auch als Mikrocode eines speziellen Computerchips

vorliegen. Dadurch ist der J/XFS Standard unabhängig von Betriebssystem und Hardware.

- Unabhängigkeit von Gerät und Hersteller
  Da J/XFS herstellerspezifische Funktionen nach Gerätetypen abstrahiert, sind J/XFS konforme Applikationen von der zugrunde liegenden Hardware unabhängig. Dies bedeutet auch, dass diese Hardware ohne Beeinträchtigung der Applikationen ersetzt oder aktualisiert werden kann.
- Ortsunabhängigkeit
  Die Kommunikationsschicht ist für den Applikationsentwickler vollkommen transparent. Es macht keinen Unterschied, ob lokal oder entfernt (remote) auf Peripheriegeräte zugegriffen wird. So ist es irrelevant, wo diese Geräte physikalisch aufgestellt sind.

KAPITEL **3**

Ein Überwachungssystem für Bankperipheriegeräte

In real life there is no <undo>!
(Roger Zacharias)

3 Ein Überwachungssystem für Bankperipheriegeräte

In der Welt der Finanzinstitute werden an Softwaresysteme besondere Anforderungen gestellt. In fachlicher Hinsicht liegt dies vor allem an der Kombination erheblicher funktionaler Komplexität mit einer ausgesprochen hohen Änderungsfrequenz, die sowohl durch die Flexibilität der Finanzmärkte als auch durch Anpassungen an gesetzliche Vorschriften bedingt ist. Wesentliche technische Merkmale dieser Systeme sind ihre extreme, oft verteilte Verfügbarkeit, Performanz und Skalierbarkeit. Heutige Softwaresysteme in diesem Bereich zeichnen sich weiterhin durch hohe laufende Kosten und eine im Verhältnis zu den Anforderungen limitierte Funktionalität aus. Globalisierung und Vernetzung, verstärkter Konkurrenzdruck und anspruchsvollere Konsumenten sind die Herausforderungen an die Finanzwirtschaft zu Beginn des 21. Jahrhunderts und damit auch an deren Softwaresysteme.

In diesem Kapitel soll, basierend auf den in *Kapitel 2 – Architekturen, Technologien und Paradigmen* gewonnenen Erkenntnissen, die Konzeption und Realisierung eines Überwachungssystem-Prototyps für Bankperipheriegeräte dargestellt werden. Zunächst wird auf den fachlichen Hintergrund und derzeitige Probleme solcher Systeme eingegangen und darauf folgend Analyse, Entwurf und Implementierung dieses Prototyps vorgestellt. Hierbei liegt der Schwerpunkt auf der Konzeption des Systems. Es soll in diesem Kapitel bewusst auf unnötig Prosa verzichtet werden, da die UML-Diagramme, nach Ansicht des Autors, aussagekräftig genug sind.

3.1 Fachlicher Hintergrund

Die traditionelle Bankfiliale befindet sich heute im vielfachen Wandel. Neue Geschäfts- und Vertriebsstrategien werden umgesetzt, Kundenorientierung, Beratung und Verkauf gewinnen in der Filiale immer mehr an Bedeutung. Gleichzeitig wächst der Kostendruck auf den Filialbetrieb. Deshalb werden Bargeldtransaktionen verstärkt auf SB-Systeme verlagert und die somit gewonnene Zeit der Bankmitarbeiter wird zunehmend für Beratung und Verkauf genutzt. Die vollständige Verlagerung aller Bargeldtransaktionen in ein reines SB-Geschäft ist heute jedoch noch nicht zu erreichen, da dem Kunden Dienstleistungen wie zum Beispiel Ein- und Auszahlungen auf bzw. von Sparbüchern, Auszahlungen von Großbeträgen usw. weiterhin angeboten werden sollen. In vielen Geschäftsstellen werden

Bankperipheriegeräte wie Geldautomaten oder Kontoauszugsdrucker aber noch nicht mit Hilfe von speziellen Management-Werkzeugen überwacht, so dass es relativ oft zu unnötigen Komplettausfällen kommt. In anderen Fällen, die sicher nicht nur den Autor in einer gewissen Weise begeistern, wird ein eventueller Fehler vom Überwachungssystem automatisch erkannt und daraufhin ein Reboot des Peripheriegerätes veranlasst, welcher in vielen Fällen ausreicht, um die Funktionstüchtigkeit des Gerätes wieder herzustellen. Eine zuverlässige SB-Hard- und Software, frühzeitiges Erkennen und Beheben von Betriebsstörungen, eine unkomplizierte Administration und Endkundenbedienbarkeit sind die Schlüsselfaktoren für den erfolgreichen Betrieb eines Selbstbedienungsnetzwerkes und unabdingbar für eine hohe Akzeptanz und Zufriedenheit sowohl bei SB-Kunden als auch bei den Betreibern dieser Systeme.

Bestehende Überwachungs- und Administrationssysteme für SB-Netzwerke erfüllen zwar zuverlässig ihren Zweck, genügen aber den weiter oben angesprochenen neuen Anforderungen nicht mehr. Insbesondere die hohen laufenden Kosten erfordern neue Konzepte in diesem Umfeld. Zum Beispiel ist die Geschäftslogik auf den Endgeräten gelagert und nicht zentral implementiert. Änderungen sind somit auf jedem Endgerät erneut durchzuführen, welches einen zeitaufwendigen und kostenintensiven Prozess darstellt. Gefragt sind flexible, dynamische, skalierbare, integrierte und web-fähige Managementsysteme, welche zusätzlich die bereits getätigten hohen Investitionen in vorhandene Systeme schützen. Ein erster Schritt in diese Richtung soll der in den folgenden Kapiteln vorgestellte Prototyp sein.

3.1.1 Management von und durch Web Services

Zunächst soll erläutert werden, wie sich die in *Kapitel 2 – Architekturen, Technologien und Paradigmen* vorgestellten Web Service- und Management-Architektur sinnvoll kombinieren lassen. Grundsätzlich existieren hierbei zwei Möglichkeiten, welche in *Abbildung 3-2 Überwachung durch Web Services* und *Abbildung 3-3 Überwachung von Web Services* dargestellt sind:

- Überwachung von Web Services

  Spätestens wenn Web Services sich durchgesetzt haben und auch unternehmenskritische Aufgaben erfüllen, ist es notwendig, diese durch spezielle Managementsysteme zu überwachen. Ein solches Managementsystem wäre in der Lage Existenz, Verfügbarkeit, Status und Performance eines Web Services zu

überwachen. Natürlich ist es zusätzlich notwendig die Web Service Infrastruktur und die hinter den Web Services liegende Geschäftslogik zu überwachen. Die benötigte Netzwerkinfrastruktur könnte in diesem Fall wie bisher durch bereits vorhandene Netzwerkmanagementprodukte überwacht werden, um auch die Basis, das Netz, abzusichern. Andere Aufgaben eines solchen Managementsystems wären der Schutz der Dienste vor unerlaubtem Zugriff durch Authentifikationsmechanismen, Verschlüsselung ein- und ausgehender Nachrichten oder einfache Logging-Mechanismen. Natürlich könnten durch ein solches System auch Nutzer und Nutzungsdauer bestimmt werden, welche zur Abrechnung der Dienstnutzung dienen.

- Überwachung durch Web Services

  Der umgekehrte Weg, d.h. die Überwachung durch Web Services, stellt eine Möglichkeit dar, Managementfunktionalität interoperabel anzubieten, erlaubt eine beliebige Kombination dieser Managementfunktionalität inklusive dem unproblematischen Aufbau von Prozessketten sowie weitere, bereits in *Kapitel 2.1.1 – Ein neues Paradigma* aufgeführte, Vorteile. Diese Variante kommt bei dem in den folgenden Kapiteln vorgestellten Prototyp zum Tragen. Eine interessante Anwendung wäre sicher die Überwachung von Web Services durch Web Services, welche zum Beispiel in folgendem Szenario sinnvoll ist: Zwei Unternehmen A und B, welche Bestandteile einer Supply Chain sind, tauschen ihre Informationen über Web Services aus. Unternehmen B hat bestimmte Web Services, welche A anbietet in eigene Applikationen integriert, die zum Beispiel dem Anstoß weiterer Web Services des Zulieferunternehmens C dienen. Sind die Dienste von A zeitweise nicht verfügbar oder überlastet, bedeutet dies für Applikation von B automatisch eine Ausnahmesituation. Sinnvoller wäre es, wenn A das Management seiner Dienste als Web Service offen legt und damit B die Möglichkeit bietet, detaillierte Informationen über den Ausfall oder die Überlastung zu erhalten um entsprechend darauf reagieren zu können. Dieses Szenario, in welchem B vor jeder Nutzung eines Web Services von A bei dessen Management Web Service dessen Verfügbarkeit erfragt, ist in *Abbildung 3-1 Offenlegung einer Managementschnittstelle als Web Service* dargestellt.

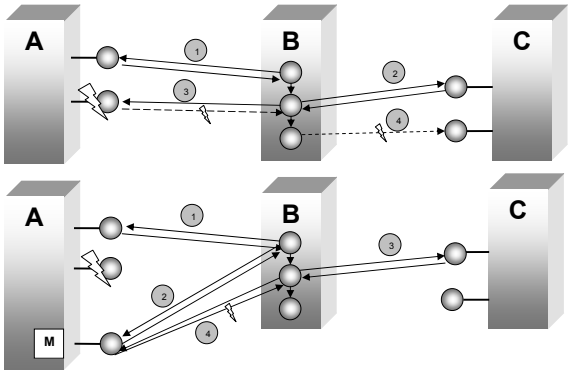

Abbildung 3-1 Offenlegung einer Managementschnittstelle als Web Service

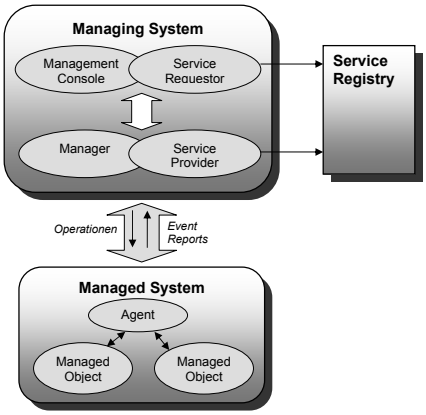

Abbildung 3-2 Überwachung durch Web Services

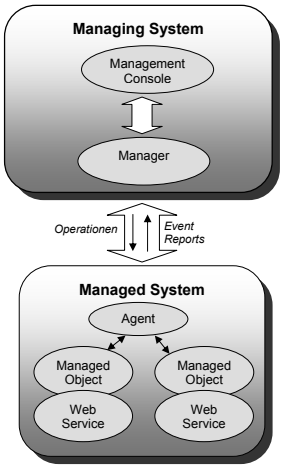

Abbildung 3-3 Überwachung von Web Services

3.2 Anforderungsanalyse

Durch die Anforderungsanalyse sollen Funktionen und Eigenschaften des Produktes, in diesem Fall des Prototypen, soweit analysiert werden, dass eine Basis für die Realisierung des Produktes gefunden wird. Hierzu werden zunächst die nichtfunktionalen und funktionalen Anforderungen bestimmt. Der zweite Schritt führt zu einer Spezifikation des Systems aus fachlicher Sicht inklusive eines Dictionaries, in welchem die Begriffe des Anwendungsgebietes erläutert werden. Der letzte und wichtigste Schritt der Anforderungsanalyse ist die Darstellung der benötigten Anwendungsfälle und der daran beteiligten Akteure. Die in *Kapitel 2 – Architekturen, Technologien und Paradigmen* beschriebenen Architekturen und die in *Kapitel 3.1 – Fachlicher Hintergrund* beschriebene Kombination dieser, legt den Grundstein für den zu erstellenden Prototyp. Das heißt, dass die Basiskomponenten und die grundlegende Funktionsweise des Prototyps bereits durch diese gegeben sind und somit damit vorausgesetzt werden.

3.2.1 Systemspezifikation

Der zu erstellende Prototyp, hat, wie in *Abbildung 3-4 Prototyp - Systemübersicht* dargestellt, drei Hauptbestandteile (wobei auf Fragen des Deployments später eingegangen werden soll):

- Die grafische Benutzeroberfläche, welche als Management Console fungiert und nach dem Web Service Modell die Rolle des Service Requestors einnimmt. Mit dieser GUI arbeitet das Überwachungspersonal. Es können Kommandos an das Gerät abgesetzt und Ereignisse vom Gerät empfangen werden.
- Einen Web Application Server, der als Middleware zwischen GUI und Peripheriegerät gelagert wird. Er fungiert als Manager und bietet die Managementschnittstelle als Web Service an, um sie beliebig verfügbar zu machen. Somit nimmt er nach dem Web Services Modell die Rolle des Service Providers ein.
- Das Peripheriegerät, welches zu überwachen ist. Im Prototyp handelt es sich hierbei um einen Geldautomaten, aber das Modell soll auf einfache Art und Weise auf beliebige Bank- und Einzelhandelsperipheriegeräte erweiterbar sein. Auf dem Peripheriegerät lagert der Agent und die von diesem überwachten Managed Objects, welche die Ressourcen dieses Gerätes repräsentieren.

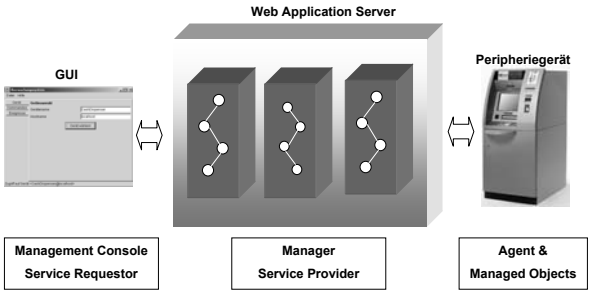

Abbildung 3-4 Prototyp – Systemübersicht

3.2.2 Nichtfunktionale Anforderungen

Folgende nichtfunktionale Anforderungen bestehen an den Prototypen:

- Entwurfseinschränkungen
  Vor Projektbeginn wurden dem Autor (aus verschiedenen Gründen) folgende Entwurfsbeschränkungen vorgegeben:
  - Als Web Application Server soll ein Servlet/JSP-konformer Applikationsserver verwendet werden.
  - Die zu verwendende Managementarchitektur soll auf dem Java Management Extensions (JMX) Standard basieren.
- Hardware
  An die Hardware der Systembestandteile werden keine erhöhten Anforderungen gestellt. Bedingung ist hier aber jeweils die Unterstützung der Internet-Infrastruktur.
- Software
  Auch an die Software werden keine erhöhten Anforderungen gestellt. Sie soll einem prototypischen Entwurf angemessen sein.
- Portabilität
  Alle Systembestandteile (GUI, Geschäftslogik und Agentenlogik) sollen portabel

sein. Die Geschäftslogik soll zudem unabhängig vom zugrunde liegenden Applikationsserver sein.

- Bedienbarkeit
  Der Prototyp soll einfach zu bedienen und zu installieren sein, um eine unkomplizierte Weiterentwicklung zu ermöglichen.

An folgende, nichtfunktionale Anforderungen bestehen, aufgrund der prototypischen Entwicklung, keine speziellen Anforderungen:

- Ausfallsicherheit
- Performance und Laufzeitverhalten
- Sicherheit
- Verfügbarkeit
- Stabilität
- Anpassbarkeit

3.2.3 Funktionale Anforderungen

Die grundlegenden funktionalen Anforderungen ergeben sich aus den in *Kapitel 2 – Architekturen, Technologien und Paradigmen* bereits ausreichend beschriebenen Anforderungen an Management-, Web Service- und Bankperipheriezugriffs-Architekturen. Folgende erweiterte funktionale Anforderungen bestehen an den Prototypen:

- Tracing und Logging
  Es sollen nur grundlegende Tracing- und Logging-Funktionalitäten auf Basis der Kommandozeile bestehen.
- Technische Dokumentation
- Installationsanleitung

Folgende funktionale Anforderungen sollen bei diesem Prototypen noch nicht realisiert werden:

- Internationalisierung
- Hilfesystem
- Benutzerhandbuch und Online-Dokumentation

3.2.4 Akteure und Anwendungsfälle

Die folgenden Abbildungen und Tabellen stellen, in systematischer Art und Weise, die vom System zu erbringenden Anwendungsfälle dar. *Abbildung 3-5 Prototyp – Use Case Diagramm* stellt zunächst das, das gesamte System umfassende, Use Case Diagramm dar. Darauf folgen die Use Case Spezifikationen der einzelnen Use Cases und zur Verdeutlichung des Ablaufs jeweils ein zugeordnetes Aktivitätsdiagramm.

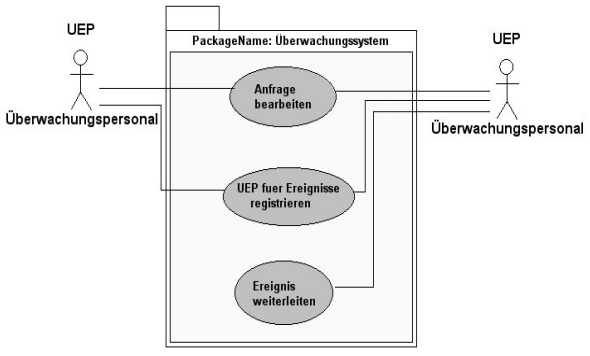

Abbildung 3-5 Prototyp - Use Case Diagramm

Wie zu sehen ist, handelt es sich lediglich um drei Anwendungsfälle. Zunächst kann das Überwachungspersonal Anfragen an das System, bzw. im engeren Sinne an das Bankperipheriegerät, stellen (z.B. Statusinformationsanfrage), auf welche es eine sofortige Antwort erhält. Weiterhin kann es sich für Ereignisse, welche aufgrund von systeminternen Änderungen (z.B. Schwellwertunterschreitungen) auftreten, registrieren. Tritt solch ein Ereignis auf, wird es automatisch an das Überwachungspersonal weitergeleitet, sofern dieses dafür registriert ist.

3.2.4.1 Use Case "Anfrage bearbeiten"

Use Case Spezifikation	
Use Case:	**"Anfrage bearbeiten" Use Case Nr.: 1**
Ziel	Das Überwachungspersonal (UEP) erhält eine detaillierte Antwort auf die gesendete Anfrage (z.B. Informationen über die Kassetten des Automaten)
Vorbedingung	keine
Nachbedingung (Erfolg)	Das UEP erhält eine detaillierte Antwort auf die gesendete Anfrage
Nachbedingung (Fehlschlag)	Das UEP erhält eine Fehlermeldung mit der Fehlerursache
Akteure	UEP = Überwachungspersonal
Auslösendes Ereignis	UEP initiiert Vorgang durch Anfrage an das System
Beschreibung	
Standardfall	Die vom Überwachungspersonal abgesetzte Anfrage (z.B.: "Gib mir detaillierte Informationen über die Kassetten des Geldautomaten") wird vom System entgegengenommen und beantwortet (Ergebnis oder Fehlermeldung)
Erweiterungen	keine Erweiterungen des Standardfalls
Alternativen	keine Alternativen zum Standardfall
Ablaufbeschreibung	siehe Tabelle und Aktivitätsdiagramm

Tabelle 3-1 Prototyp – Use Case Nr.1 Spezifikation

Use Case-Ablaufbeschreibung		
Use Case:	**"Anfrage bearbeiten" Use Case Nr.: 1**	
Ablauf:		
Nr.	**Zuständigkeit**	**Beschreibung**
1		Erfassung und Prüfung der Anfragedaten
1.1	UEP	Die Verbindungsdaten (IP, Gerätename) werden eingegeben
1.2	System	Es wird geprüft, ob die Verbindungsdaten gültig sind
1.3	System	Ausnahme: Verbindungsdaten ungültig
1.4	UEP	Die gewünschte Anfrage wird aus einer Liste ausgewählt
1.5	System	Es wird geprüft, ob die gewünschte Anfrage ausführbar ist (z.B. ist die Kassettenzählerabfrage bei einem Kontoauszugsdrucker nicht ausführbar)
1.6	System	Ausnahme: Anfrage nicht ausführbar
2		Anfrage bearbeiten
2.1	System	Anfrage wird vom System bearbeitet
2.2	System	Ausnahme: Fehler bei der Bearbeitung
2.3	System	Es wird eine Antwort generiert
2.4	System	Die Antwort wird dem UEP zugestellt
Ausnahmen:		
Nr.	**Zuständigkeit**	**Beschreibung**
1.3		Verbindungsdaten ungültig
1.3.1	System	System meldet Ungültigkeit
1.3.2	UEP	Neueingabe durch UEP
1.6		Anfrage nicht ausführbar
1.6.1	System	System meldet Nichtausführbarkeit
1.6.2	UEP	Neuauswahl durch UEP
2.2		Fehler bei der Bearbeitung
2.2.1	System	generierte Antwort enthält Beschreibung des Bearbeitungsfehlers

Tabelle 3-2 Prototyp – Use Case Nr.1 Ablaufbeschreibung

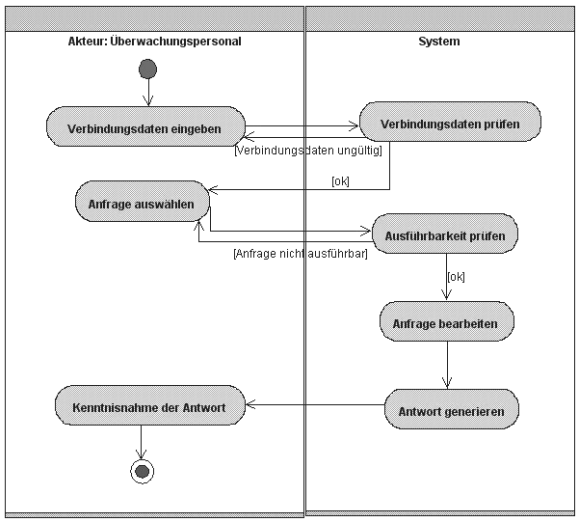

Abbildung 3-6 Prototyp – Use Case Nr.1 Aktivitätsdiagramm

3.2.4.2 Use Case "Überwachungspersonal für Ereignisse registrieren"

Use Case Spezifikation	
Use Case:	**"UEP für Ereignisse registrieren" Use Case Nr.: 2**
Ziel	Das System kann jedes registrierte UEP über auftretende Ereignisse informieren (siehe Use Case Nr. 3 - "Ereignis weiterleiten")
Vorbedingung	keine
Nachbedingung (Erfolg)	Das UEP erhält eine Bestätigung für die Registrierung
Nachbedingung (Fehlschlag)	Das UEP erhält eine Fehlermeldung mit der Fehlerursache
Akteure	UEP = Überwachungspersonal
Auslösendes Ereignis	UEP initiiert Vorgang durch Anfrage an das System
Beschreibung	
Standardfall	Die vom Überwachungspersonal abgesetzte Registrierungs-Anfrage wird vom System entgegengenommen und beantwortet (Ergebnis oder Fehlermeldung)
Erweiterungen	keine Erweiterungen des Standardfalls
Alternativen	keine Alternativen zum Standardfall
Ablaufbeschreibung	siehe Tabelle und Aktivitätsdiagramm

Tabelle 3-3 Prototyp – Use Case Nr.2 Spezifikation

Use Case Ablaufbeschreibung		
Use Case:	**"UEP für Ereignisse registrieren" Use Case Nr.: 2**	
Ablauf:		
Nr.	**Zuständigkeit**	**Beschreibung**
1		Erfassung und Prüfung der Registrierungsdaten
1.1	UEP	Die Verbindungsdaten (IP, Gerätename) werden eingegeben
1.2	System	Es wird geprüft, ob die Verbindungsdaten gültig sind
1.3	System	Ausnahme: Verbindungsdaten ungültig
2		UEP für Ereignisse registrieren
2.1	System	Der UEP wird der Liste der Ereignisempfänger hinzugefügt
2.2	System	System stellt dem UEP eine Bestätigung zu
Ausnahmen:		
Nr.	**Zuständigkeit**	**Beschreibung**
1.3		Verbindungsdaten ungültig
1.3.1	System	System meldet Ungültigkeit
1.3.2	UEP	Neueingabe durch UEP

Tabelle 3-4 Prototyp – Use Case Nr.2 Ablaufbeschreibung

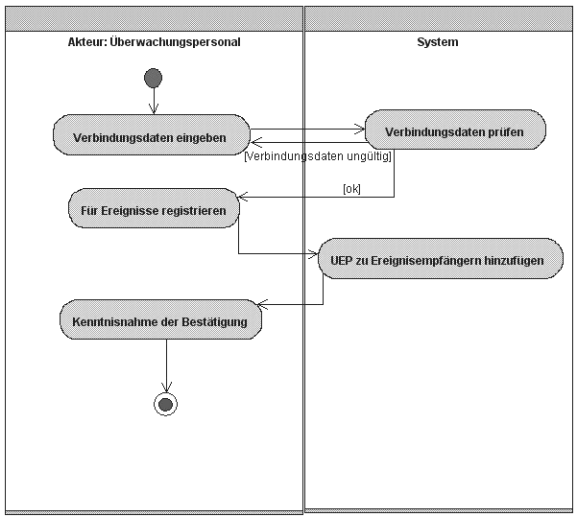

Abbildung 3-7 Prototyp – Use Case Nr.2 Aktivitätsdiagramm

3.2.4.3 Use Case "Ereignis weiterleiten"

Use Case Spezifikation	
Use Case:	**"Ereignis weiterleiten" Use Case Nr.: 3**
Ziel	Auf dem Peripheriegerät auftretende Ereignisse (s.u.) werden dem Überwachungspersonal mitgeteilt.
Vorbedingung	Use Case "Kunde für Ereignisse registrieren"
Nachbedingung (Erfolg)	Das UEP erhält detaillierte Informationen über das aufgetretene Ereignis
Nachbedingung (Fehlschlag)	Das UEP erhält eine Fehlermeldung mit der Fehlerursache
Akteure	UEP = Überwachungspersonal
Auslösendes Ereignis	• Fehler (z.B. Hardwarefehler) • Probleme (z.B. Geldkassette leer) • Interaktion mit Gerätebenutzer (z.B. Karte wurde einbehalten) • Interaktion mit Operator (z.B. Gerät wird gerade repariert)
Beschreibung	
Standardfall	Auf dem Peripheriegerät auftretende Ereignisse (s.o.) werden dem Überwachungspersonal mitgeteilt.
Erweiterungen	keine Erweiterungen des Standardfalls
Alternativen	keine Alternativen zum Standardfall
Ablaufbeschreibung	siehe Tabelle und Aktivitätsdiagramm

Tabelle 3-5 Prototyp – Use Case Nr.3 Spezifikation

Use Case Ablaufbeschreibung		
Use Case:	**"Ereignis weiterleiten" Use Case Nr.: 3**	
Ablauf:		
Nr.	**Zuständigkeit**	**Beschreibung**
1		Ereignis aufnehmen
1.1	System	System nimmt Ereignis entgegen
1.2	System	Das Ereignis wird gespeichert
2		Ereignismeldung
2.1	System	Ereignismeldung generieren
2.2	System	System stellt dem UEP die Ereignismeldung zu
Ausnahmen:		
Nr.	**Zuständigkeit**	**Beschreibung**
-	-	-

Tabelle 3-6 Prototyp – Use Case Nr.3 Ablaufbeschreibung

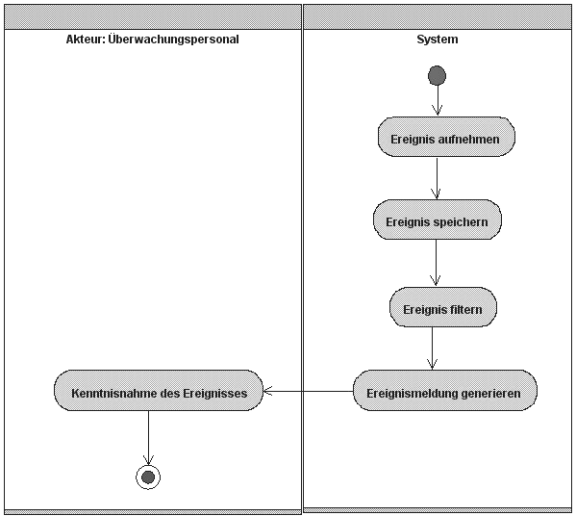

Abbildung 3-8 Prototyp – Use Case Nr.3 Aktivitätsdiagramm

3.3 Konzeptionelles Modell

Im Konzeptionellen Modell werden die fachlichen Konzepte des Systems auf einem abstrakten Niveau, aber unabhängig von der Implementierung, beschrieben. Dies umfasst die Statik, die Dynamik und das Verhalten des Systems, wofür in der UML entsprechende Notationskonstrukte existieren. Diese Sichtweise ist Black-Box-artig, da an dieser Stelle nur wichtig ist, welche Komponenten und Beziehungen im betrachteten System vorkommen, aber nicht, wie diese sich zusammensetzen.

3.3.1 Statik des Systems

Die Statik des Systems wird durch Klassendiagramme beschrieben. Zusätzlich ist es sinnvoll ein Fachlexikon einzubringen, in welchem die wichtigsten Begriffe des Klassendiagramms und des Systemumfeldes näher erläutert werden. Beides wird für den vorgestellten Prototyp auf den folgenden Seiten dargestellt.

3.3.1.1 Klassendiagramm

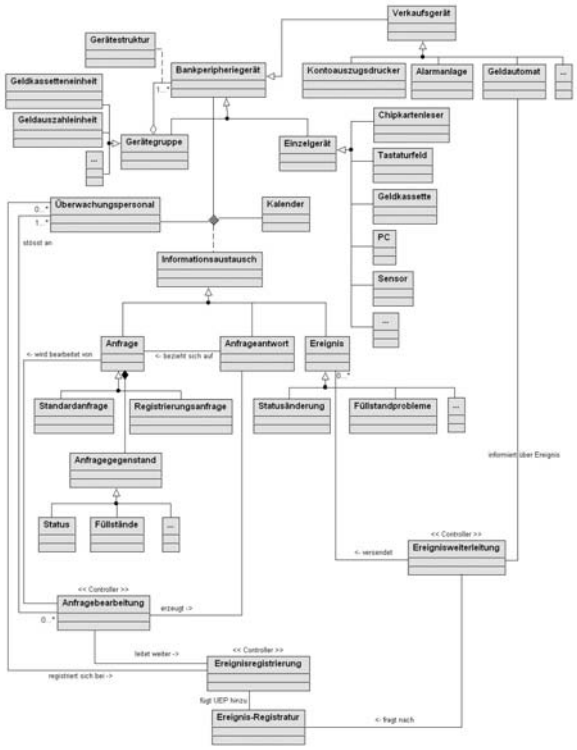

Abbildung 3-9 Prototyp – Klassendiagramm

3.3.1.2 Fachlexikon

Klasse	**Bankperipheriegerät**
Beschreibung	Diese Klasse repräsentiert ein real existierendes Bankperipheriegerät (vom Kartenlesegerät bis hin zum vollständigen Geldautomaten)
Verantwortlichkeit	• Repräsentation des realen Gerätes • Verwaltung aller Eigenschaften, Fähigkeiten und Stati des Gerätes • Verwaltung aller untergeordneten Geräte • Ereignisinteressierter kann sich registrieren und deregistrieren • Aufgetretene Ereignisse werden an alle Ereignisinteressierten versendet
Attribute	• Gerätename (Name des Gerätes) • ID (Identifikator) • Status (aktueller Status des Gerätes) • Fähigkeiten (Liste der gerätespezifische Fähigkeiten)
Klasse	**Verkaufsgerät**
Beschreibung	Abgeleitet von: Bankperipheriegerät Integriertes verkaufsfähiges Gerät, bestehend aus einer Reihe von Komponenten (Beispiel: Geldautomat, Alarmanlage)
Verantwortlichkeit	• Verwaltung der zugehörigen Komponenten
Attribute	• Komponenten (zugehörige Komponenten)

Klasse	**Einzelgerät**
Beschreibung	Abgeleitet von: Bankperipheriegerät Bestandteil einer Gerätegruppe oder eigenständiges Einzelteil. (Beispiele: Chipkartenleser, Tastaturfeld, Geldkassette, PC, Sensor)
Verantwortlichkeit	• Siehe Bankperipheriegerät
Attribute	• Siehe Bankperipheriegerät
Klasse	**Gerätegruppe**
Beschreibung	Abgeleitet von: Bankperipheriegerät Baugruppe bestehend aus Einzelgeräten oder anderen Baugruppen. (Beispiel: Geldkassetteneinheit = {Geldkassette} + ...)
Verantwortlichkeit	• Verwaltung aller zugehöriger Komponenten
Attribute	• Komponenten (zugehörige Komponenten)

Klasse	**Überwachungspersonal (ÜEP)**
Beschreibung	Bediener des Systems, interagieren über eine grafische Benutzeroberfläche mit dem System, stellen Anfragen an das System, registrieren sich für Ereignisse und erhalten Ereignisse vom Bankperipheriegerät.
Verantwortlichkeit	• Interaktion mit dem System • Externer Systembediener
Attribute	• ID (Identifikation)
Klasse	**Informationsaustausch**
Beschreibung	Informationen, welche zwischen UEP und Bankperipheriegerät ausgetauscht werden.
Verantwortlichkeit	• Informationsträger
Attribute	• Absender (ID des UEP oder des Bankperipheriegerätes) • Empfänger (ID des UEP oder des Bankperipheriegerätes) • Art (Art des Informationsaustausches) • Inhalt (Informationsinhalt)

Klasse	**Anfrage**
Beschreibung	Abgeleitet von: Informationsaustausch Stellt eine Anfrage des UEP an das Bankperipheriegerät dar. Beispiel: „Liefere den aktuellen Status!“
Verantwortlichkeit	• Repräsentation einer Anfrage des UEP an das Bankperipheriegerät
Attribute	• Anfragegegenstand (Ziel und Sinn der Anfrage)
Klasse	**Anfrageantwort**
Beschreibung	Abgeleitet von: Informationsaustausch Stellt eine direkte Antwort des Bankperipheriegeräts auf eine Anfrage des UEP dar. Beispiel: Statusinformationen
Verantwortlichkeit	• Repräsentation der Antwort des Bankperipheriegeräts auf eine Anfrage des UEP
Attribute	• Antwort (Antwort auf die eingegangene Anfrage)

Klasse	**Ereignis**
Beschreibung	Abgeleitet von: Informationsaustausch Stellt ein auf dem Bankperipheriegerät aufgetretenes Ereignis dar. Beispiel: Statusänderungen, Füllstandsprobleme
Verantwortlichkeit	• Repräsentation eines auf dem Bankperipheriegeräts aufgetretenen Ereignisses
Attribute	• Typ (Ereignistyp) • Informationen (Detailinformationen über Ereignis)
Klasse	**Ereignis-Registratur**
Beschreibung	Registratur (Registry), welche alle ereignisinteressierten UEP verwaltet.
Verantwortlichkeit	• Verwaltung aller registrierten ereignisinteressierten UEP • Annahme neuer ereignisinteressierter UEP • Weitergabe der Registraturinformationen an die Ereignisweiterleitung
Attribute	• Ereignisinteressierte (Liste der Ereignisinteressierten)

Klasse	**Anfragebearbeitung**
Beschreibung	<<Controller-Klasse>> Nimmt Anfragen des UEP an, delegiert Anfrage, erzeugt Anfrageantwort und sendet diese zurück an UEP.
Verantwortlichkeit	• Anfragen vom UEP entgegennehmen • Anfragen an Bankperipheriegerät delegieren • Antwort vom Bankperipheriegerät entgegennehmen • Antwort an UEP generieren • Antwort an UEP senden
Attribute	
Klasse	**Ereignisregistrierung**
Beschreibung	<<Controller-Klasse>> Nimmt Ereignisregistraturanfragen vom UEP an und fügt diese der Ereignisregistratur hinzu.
Verantwortlichkeit	• Ereignisregistraturanfragen vom UEP entgegennehmen • UEP in die Ereignisregistratur hinzufügen
Attribute	

Klasse	**Ereignisweiterleitung**
Beschreibung	<<Controller-Klasse>> Leitet aufgetretene Ereignisse an alle ereignisinteressierten UEP weiter.
Verantwortlichkeit	• Entgegennahme der Ereignisse vom Bankperipheriegerät • Extrahieren der Ereignisinteressierten aus der Ereignisregistratur • Versenden der Ereignisse an alle Ereignisinteressierten
Attribute	

Tabelle 3-7 Prototyp – Fachlexikon

3.3.2 Dynamik des Systems

Die Dynamik des Systems wird in erster Linie mit Hilfe von Sequenzdiagrammen beschrieben. Für jeden der drei Use Cases (siehe *Abbildung 3-5 Prototyp – Use Case Diagramm*) existiert eine grundlegende Befehlssequenz. Diese Befehlssequenzen bilden die Basis der Systemdynamik. Diese Befehlssequenzen sollen im Folgenden vorgestellt werden.

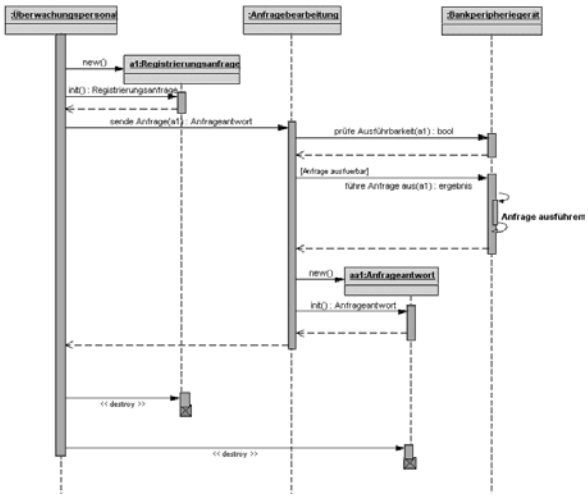

Abbildung 3-10 Prototyp - Sequenzdiagramm "Anfrage bearbeiten"

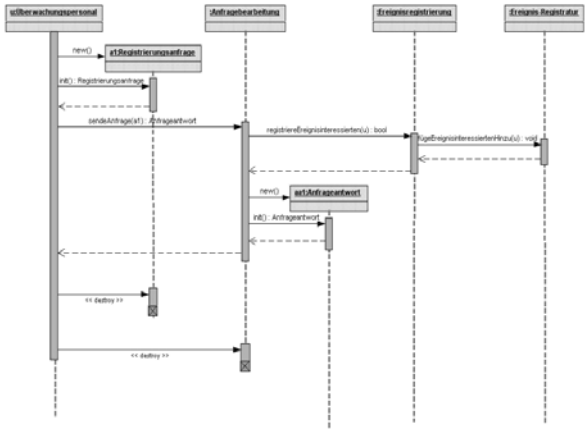

Abbildung 3-11 Prototyp - Sequenzdiagramm "Registrierung für Ereignisse"

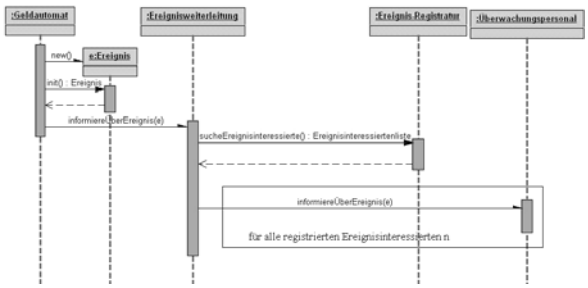

Abbildung 3-12 Prototyp - Sequenzdiagramm "Ereignis weiterleiten"

3.4 Grobentwurf (Top Level Design)

Der Grobentwurf dient der Umsetzung der fachlichen Analyseergebnisse in DV-Strukturen. Das heißt hier wird (im Rahmen der funktionalen- und nichtfunktionalen Anforderungen) entschieden, welche Programmiersprachen, Technologien, APIs, Klassenbibliotheken usw. verwendet werden und wie diese optimal kombiniert werden können. Weiterhin werden an dieser Stelle erste Entwürfe der Benutzerschnittstelle erstellt.

3.4.1 Benutzerschnittstelle

Dank der prototypischen Natur des zu erstellenden Systems kommt man an dieser Stelle mit einer sehr einfachen Benutzerschnittstelle aus, welche im Folgenden vorgestellt werden soll. Sie besteht aus folgenden Masken:

- Geräteauswahl-Maske
  Diese Maske erlaubt dem Benutzer das Gerät auszuwählen, an welches Anfragen gesendet und von welchen Ereignissen empfangen werden sollen. Hierzu muss der eindeutige Gerätename und der Hostname des Gerätes eingegeben werden. Nach Bestätigung der Auswahl wird eine Überprüfung vorgenommen, ob das gewählte Gerät existiert bzw. erreichbar ist. Ist dies nicht der Fall wird eine entsprechende Fehlermeldung ausgegeben.
- Kommando-Maske
  Die Kommando-Maske erfüllt zwei Aufgaben: Zum einen ist es dem Benutzer an dieser Stelle möglich Anfragen an das Gerät zu senden und zum anderen werden die Antworten auf diese Anfragen auf der gleichen Maske dargestellt. Der Prototyp enthält nur zwei mögliche Kommandos.
- Ereignis-Maske
  Auch diese Maske erfüllt zwei Aufgaben: Zum einen kann man sich an dieser Stelle für Ereignismeldungen vom Gerät registrieren und zum anderen werden auf dieser Maske die einkommenden Ereignisse dargestellt, sofern man sich vorher registriert hat.

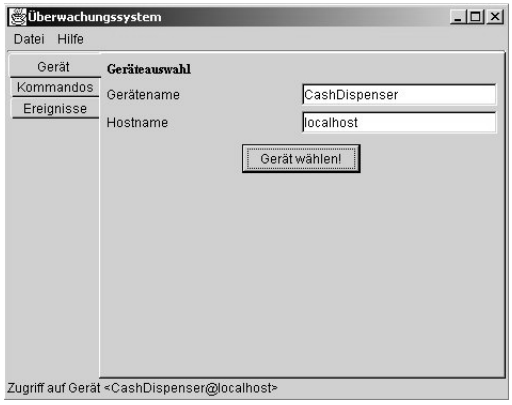

Abbildung 3-13 Prototyp - Geräteauswahl-Maske

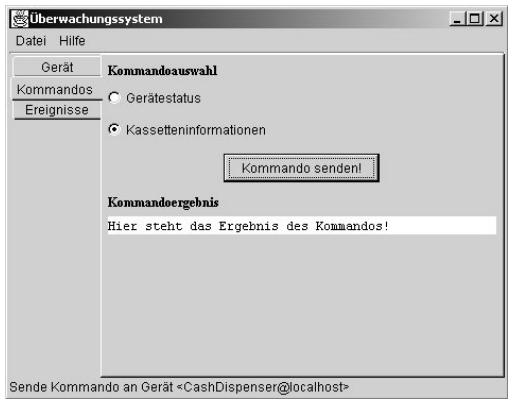

Abbildung 3-14 Prototyp - Kommando-Maske

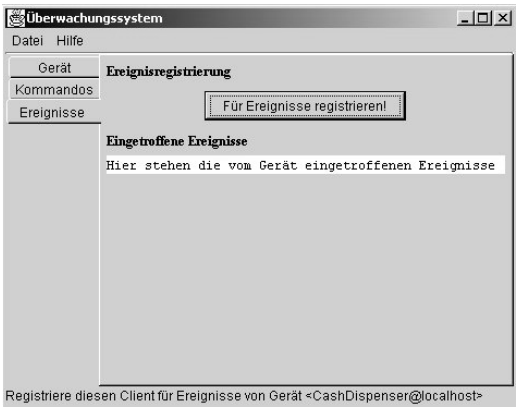

Abbildung 3-15 Prototyp - Ereignis-Maske

3.4.2 Architektur

Die Top-Level Architektur, welche den groben Aufbau des Systems beschreibt, stellt sich, wie bereits in *Kapitel 3.2.1 – Systemspezifikation* beschrieben, folgendermaßen dar:

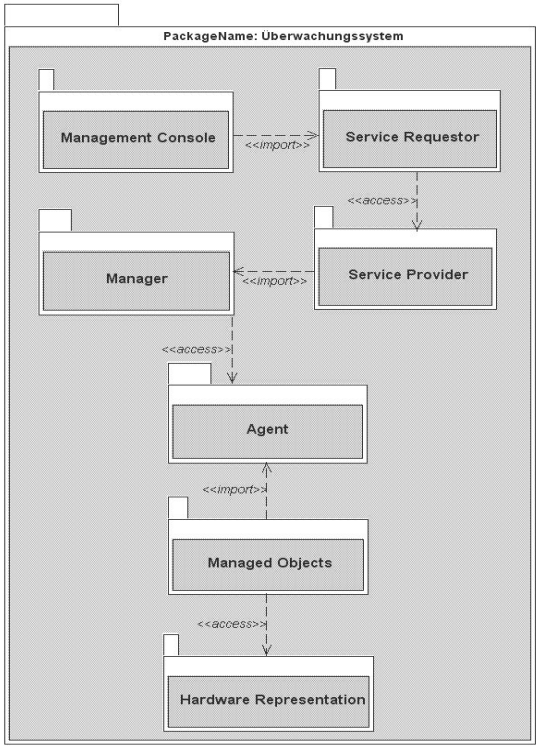

Abbildung 3-16 Prototyp - Top Level Architektur

Nach dieser Aufteilung müssen für die einzelnen Komponenten und die Kombination dieser, entsprechende Technologien, Produkte und APIs gefunden werden, womit sich dieser Prototyp realisieren lässt.

3.4.3 Technologien, Produkte und APIs

Da es sich hier um einen Prototypen handelt, wurde darauf geachtet, dass die verwendeten Technologien, Produkte und APIs allgemein anerkannten Standards entsprechen, möglichst einfach zu verwenden und zu implementieren sind und im Idealfall Referenzimplementierungen darstellen. An dieser Stelle sollen daher keine Vergleiche bezüglich Wirtschaftlichkeit, Performance, usw. zwischen einzelnen Produkten oder APIs durchgeführt werden. Wie bereits in *Kapitel 3.2.2 – Nichtfunktionale Anforderungen* angesprochen, sind folgende Rahmenbedingungen bereits vorgegeben:

- Der verwendete Middleware-Server soll ein Servlet/JSP-konformer Web Application Server sein.
- Die zu verwendende Managementarchitektur soll auf dem Java Management Extensions (JMX) Standard basieren.

Im Folgenden werden die Technologien, Produkte und APIs vorgestellt, auf welche die Entscheidung gefallen ist:

- J2SE 1.4.0 beta 2
  Als Plattform sowohl für die Entwicklung als auch als Laufzeitumgebung für die einzelnen Komponenten wurde die aktuelle Java 2 Standard Edition (J2SE) gewählt. Weitere Informationen finden sich unter http://java.sun.com/j2se.
- Apache Tomcat 4.01
  Hierbei handelt es sich um einen HTTP-Server und JSP/Servlet-Container, welcher die offizielle Referenzimplementierung der aktuellen Java Servlet 2.3 Technologie und der JavaServer Pages 1.2 Technologie darstellt. Weitere Informationen finden sich unter http://jakarta.apache.org/tomcat.
- Apache SOAP 2.2
  Dieses Open-Source Produkt der Apache Group ist die zurzeit im Java Bereich am weitesten verbreitete Implementierung der SOAP v1.1 und der SOAP Messages with Attachments Spezifikationen. Auf Apache SOAP wird weiter unten

  innerhalb des *Kapitels 3.4.3.1 - Apache SOAP 2.2* näher eingegangen. Weitere Informationen finden sich unter http://xml.apache.org/soap.
- Apache Axis 1.0 alpha 2 Nightly Build
  Hierbei handelt es sich um den Nachfolger des Apache SOAP 2.2 Projektes, d.h. Apache Axis ist eigentlich Apache SOAP 3.0. Auch dieses Produkt stellt eine Implementierung der SOAP v1.1 und der SOAP Messages with Attachments Spezifikationen dar, zeichnet sich jedoch durch eine höhere Performance, verbesserte Flexibilität und Modularität und insbesondere durch bessere Zusammenarbeit mit anderen SOAP-Implementierungen (z.B. Microsoft SOAP) aus. Auf Apache Axis wird weiter unten innerhalb des *Kapitels 3.4.3.2 – Apache Axis 1.0 alpha 2 Nightly Build* näher eingegangen. Weitere Informationen finden sich unter http://xml.apache.org/axis.
- JMX 1.0 Spezifikation
  Hierbei handelt es sich um ein von Sun Microsystems und Partnern entwickeltes Management-Framework der ´dritten Generation´. Es wird in *Kapitel 2.2.6 JMX – Die Management-Architektur von Morgen?* detailliert erläutert. Weitere Informationen finden sich unter http://java.sun.com/products/JavaManagement.
- JDMK 4.2
  Hierbei handelt es sich um die kommerzielle Referenzimplementierung der JMX 1.0 Spezifikation. Diese wird von Sun vertrieben und ist neben einer IBM Tivoli Lösung zurzeit die einzige vollständig standardkonforme Implementierung. Weitere Informationen finden sich unter http://java.sun.com/products/jdmk.
- J/XFS 2.0
  Die Extensions for Financial Services for the Java Platform (J/XFS) sind ein Standard zum Zugriff auf Bank- und Einzelhandelperipheriegeräte. Dieser Standard wird in *Kapitel 2.3 – Technologien zum Zugriff auf Bankperipheriegeräte* detailliert erläutert. Weitere Informationen finden sich unter http://www.jxfs.com.
- FDI for J/XFS 2.0
  Das Financial Device Interface (FDI) for J/XFS stellt eine erweiterte Referenz-implementierung des J/XFS Standards dar. Dieses Produkt ist vom J/XFS Forum kostenlos erhältlich. Weitere Informationen finden sich unter http://www.jxfs.com.
- Xerces 1.4.3
  Hierbei handelt es sich um den wohl am weitesten verbreiten XML Parser. Dieses

  Open-Source Produkt der Apache Group ist für die Programmiersprachen Java, C++ und Perl erhältlich. Zur Zeit wird an einer zweiten Version gearbeitet, die aber an dieser Stelle aus Kompatibilitätsgründen noch nicht zum Einsatz kommen soll. XML Parser werden in *Kapitel 2.1.2.3.3 – Programmierschnittstellen und Parser* detailliert erläutert. Weitere Informationen finden sich unter http://xml.apache.org/xerces-j.
- UDDI4J 2.0 beta
  Diese, unter der Schirmherrschaft von IBM entwickelte, Open-Source Java API und Referenzimplementierung ermöglicht den programmatischen Zugriff auf eine UDDI Registry. Somit ist es auf einfache Art und Weise möglich, bereitgestellte oder unternehmensinterne Service Registries zu verwenden, um eigene Dienste bereitzustellen oder fremde Dienste aufzufinden. Weitere Informationen finden sich unter http://oss.software.ibm.com/developerworks/opensource/uddi4j.
- Castor 0.93
  Castor ist ein weit verbreitetes Open-Source Databinding Framework für Java, welches es ermöglicht Java Objekte in XML-Dokumente und XML-Dokumente in Java Objekte umzuwandeln. Dieses Framework wird für die SOAP-Kommunikation mit komplexen Objekten benötigt. Der Autor erwartet aber, dass ein solches Framework in den nächsten SOAP-Implementierungen bereits enthalten ist. Auf die Databinding Technik und das Castor Framework wird weiter unten innerhalb des *Kapitels 3.4.3.4 – Castor 0.9.3* näher eingegangen. Weitere Informationen finden sich unter http://castor.exolab.org.
- Microsoft .NET Framework
  Das .NET Framework dient in erster Linie der Entwicklung von und als Laufzeitumgebung für Web Services. Dieses neue Microsoft Framework wird in *Kapitel 2.1.6.3 – Microsoft .NET* detailliert erläutert. Weitere Informationen unter http://www.microsoft.com/net.

3.4.3.1 Apache SOAP 2.2

Apache SOAP ist eine Open-Source Java-Implementierung der SOAP v1.1 und SOAP Messages with Attachment Spezifikationen. Dieses Framework bietet:

- eine Client-seitige API zum Zugriff auf beliebige Web Services, d.h. die Möglichkeit SOAP-Nachrichten zu senden und zu empfangen
- eine Server-seitige Infrastruktur, um Web Services zu hosten
- RPC und Messaging via HTTP und SMTP

Der grundlegende Aufbau stellt sich folgendermaßen dar:

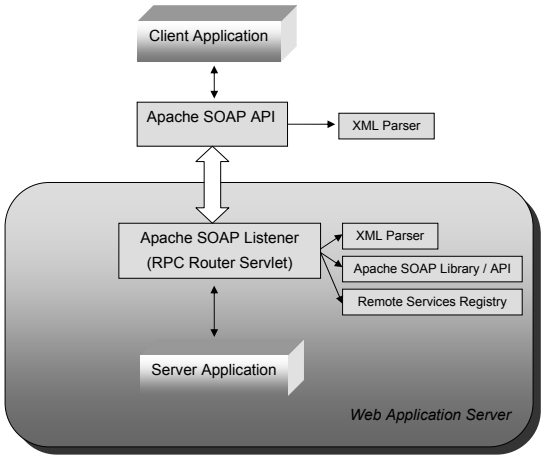

Abbildung 3-17 Apache SOAP Framework

Apache SOAP ermöglicht somit die Erstellung und Benutzung von Web Services auf Java Basis. Beide Prozesse sind, aus Sicht des Autors, intuitiv und einfach gestaltet, wodurch eine problemlose Entwicklung ermöglicht wird.[1] Zur Erstellung eines SOAP

[1] Noch intuitiver ist jedoch die Entwicklung mit Microsoft Visual Studio .NET (siehe *Kapitel 2.1.6.3 – Microsoft .NET*).

Clients werden Klassen der Apache SOAP API genutzt (siehe *Abbildung 3-19 Einfacher Apache SOAP Client*). Ein Web Service kann eine einfache Java-Klasse ohne Kenntnis des Frameworks sein. Diese muss aber deployed werden, damit das Framework Kenntnis von dieser hat. Dieser Vorgang geschieht über entsprechende Deployment-Deskriptoren oder eine grafische Administrator-Oberfläche. Ist dies geschehen, kann von außen auf den Web Service zugegriffen werden, wobei ein Servlet alle eingehenden Anfragen entgegennimmt und an den entsprechenden Web Service weiterleitet.

Folgende Abbildungen sollen die Funktionsweise dieses Frameworks verdeutlichen. In der ersten Abbildung wird ein einfacher Web Service dargestellt, der einen eingegangen String in Grossbuchstaben umwandelt, die zweite stellt einen Client vor, welcher diesen Web Service nutzt.

```
public class RzSimpleWebService {
    public String getTestString(String input) {
        return input.toUpperCase();
    }
}
```

Abbildung 3-18 Einfacher Web Service

```
public void invokeWebService() {
        URL    soapServerUrl  = new URL("http://localhost:8080/soap/servlet/rpcrouter");
        String  webServiceURI = "urn:rzsimplewebservice";
        String  methodName    = "getTestString";
        String  param         = "test";
        Vector params = new Vector();
        params.addElement(new Parameter("input", String.class, param, null));

        // build call
        Call call = new Call();
        call.setTargetObjectURI(webServiceURI);
        call.setMethodName(methodName);
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC); // response style
        call.setParams(params);

        // invoke call
        Response response = call.invoke(soapServerUrl, "");
        Parameter result = resp.getReturnValue();
        String res = (String) result;
}
```

Abbildung 3-19 Einfacher Apache SOAP Client

3.4.3.2 Apache Axis 1.0 alpha 2 Nightly Build

Axis stellt die ´dritte Generation´ des Apache SOAP Toolkits dar. Es ist zurzeit nur in der zweiten Alpha Version verfügbar, wird aber permanent weiterentwickelt. Dieses Framework zeichnet sich gegenüber dem Vorgänger durch höhere Performance, verbesserte Flexibilität und Modularität, einfacheres Deployment, eine einfachere API und insbesondere durch bessere Zusammenarbeit mit anderen SOAP-Implementierungen (z.B. Microsoft SOAP) aus. Der wichtigste Unterschied zum Vorgänger ist aber die WSDL-Unterstützung. So ist es möglich durch einfaches Anhängen von ´?wsdl´ an die URL des Web Services die WSDL für diesen Web Service zu erhalten. Diese kann nun wiederum dazu benutzt werden einen Client-Proxy zum vereinfachten Zugriff auf den Web Service zu erzeugen (siehe *Kapitel 2.1.4 – Web Service Description Language*). Des Weiteren sind Werkzeuge zur Generierung von Stubs und Skeletons aller Art und eine Reihe bereits deployter Zusatzdienste enthalten. Apache Axis wurde, obwohl es erst in der zweiten Alpha Version verfügbar ist, wegen seiner WSDL-Unterstützung, für den zu erstellenden Prototyp benutzt. Damit ließ sich, bis auf kleinere noch bestehende Probleme, eine Anbindung eines .NET Clients an das Management-Framework und somit als ´proof of concept´ eine lose Kopplung zwischen der Microsoft- und der Java-Welt realisieren.

3.4.3.3 Castor 0.9.3

Spätestens seit dem Zeitpunkt, da XML immer größere Bedeutung erlangt hat, ist ein einfacher Mechanismus zur Konvertierung zwischen XML und den Objekten der Applikationslogik wichtig. Dieser Prozess wird als ´XML Data Binding´ bezeichnet. Er erlaubt den Entwicklern weiterhin mit den Objekten der Geschäftslogik zu arbeiten, ohne aufwendigen Low-Level Parsing-Code schreiben zu müssen. Mit Databinding Frameworks ist es möglich aus vorhandenen DTDs oder XML Schemata entsprechende Java-Klassen zu erzeugen, welche den Code für die Konvertierung zu und von XML-Dokumenten und für die Validierung bereits beinhalten. Die Konvertierung nach XML ist insbesondere dann sinnvoll, wenn Daten ausgetauscht oder gespeichert werden sollen. Die Vorteile dieser Frameworks sind die folgenden:

- Der Softwareentwickler muss durch diesen automatischen Zugriff auf die XML Daten lediglich die Applikation erstellen, welche diese Daten nutzt. Es ist nicht

  mehr notwendig aufwendigen Code zum Einlesen, Validieren und Speichern der Daten zu erstellen.
- Dadurch, dass sich die Validierungs-Constraints der Daten in der DTD bzw. im XML Schema befinden, ist der eigentliche Code einfacher wart- und handhabbar.
- Wenn sich das Datenaustauschformat und damit die DTD bzw. das XML Schema ändert, ist keine aufwendige Programmänderung nötig. Das neue Schema muss lediglich noch einmal das Framework durchlaufen.
- Durch die integrierte Lösung und die Nutzung von ´in memory objects´ anstatt des aufwendigen Parsing-Codes ist die Verarbeitungsgeschwindigkeit der Programme um ein Vielfaches höher.

Es existieren eine Reihe dieser Frameworks, wobei die meisten Einschränkungen oder Probleme unterschiedlichster Art haben: Einige benötigen die Erstellung zusätzlicher Data Binding Schema Dokumente, andere können nur DTDs umwandeln und wieder andere sind auf eine Programmiersprache beschränkt. Eine reifere Lösung findet sich im Castor Framework der Exolab Group. Castor ist ein weit verbreitetes Open-Source Databinding Framework für Java, welches es ermöglicht Java Objekte in XML Dokumente und XML Dokumente in Java Objekte umzuwandeln und hierbei sowohl mit DTDs als auch mit XML Schemata umzugehen. Die XML Dokumente sind hierbei aber nicht java-zentriert, wie dies in anderen Frameworks der Fall ist, d.h. sie können problemlos auch durch Frameworks anderer Programmiersprachen in deren Objektrepräsentation umgewandelt werden. Die Begrifflichkeiten im Bereich des Data Binding erklärt die folgende Abbildung.

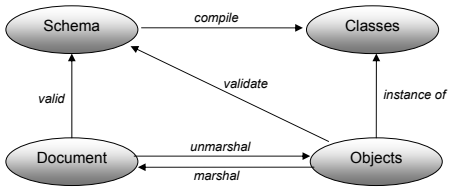

Abbildung 3-20 XML Data Binding

Die Vorgehensweise zur Benutzung dieses Frameworks sieht folgendermaßen aus:

- Zunächst muss, wenn nicht schon vorhanden, eine DTD oder ein XML Schema erstellt werden, welches die auszutauschenden oder zu speichernden Daten repräsentiert. So können zwei Partnerunternehmen nur durch Austausch dieses Dokumentes miteinander kommunizieren, ohne die verarbeitende Logik und deren Schnittstellen des Gegenübers zu kennen.
- Im nächsten Schritt wird die DTD bzw. das XML Schema dem Data Binding Framework übergeben, welches daraus entsprechende Java-Klassen generiert, welche nun nur noch kompiliert werden müssen.
- Im dritten Schritt können diese Klassen verwendet werden, um Objekte davon zu instanziieren, diese Objekte in XML Dokumente umzuwandeln (Marshalling) oder XML Dokumente einzulesen und in Java Objekte zu konvertieren (Unmarshalling). Um noch einmal auf die Unternehmen im ersten Schritt zurückzukommen: Da das Format der auszutauschenden Daten durch die DTD bzw. das XML Schema bekannt ist, kann die Geschäftslogik des einen Unternehmen zum Beispiel auf Basis der Java-Plattform und die des zweiten Unternehmens mit Hilfe der Microsoft Technologien realisiert sein. Die Daten werden über XML ausgetauscht und beim Erhalten in das entsprechende Format der Applikationslogik umgewandelt.

Dieses Framework wird für die SOAP-Kommunikation mit komplexen Objekten benötigt, da SOAP zurzeit nur einfache Datentypen und Felder von einfachen Datentypen unterstützt. Der Autor erwartet aber, dass ein solches Framework in den nächsten SOAP-Implementierungen bereits enthalten sein wird. Im entwickelten Prototyp wird Castor an beiden Seiten der SOAP-Verbindung genutzt.

Mit Hilfe der beschriebenen Technologien, Produkte und APIs stellt sich die Realisierung der Top Level Architektur aus *Abbildung 3-16 Top Level Architektur* folgendermaßen dar (in Klammern ist jeweils die entsprechend verwendete Technologie für dieses Package angegeben):

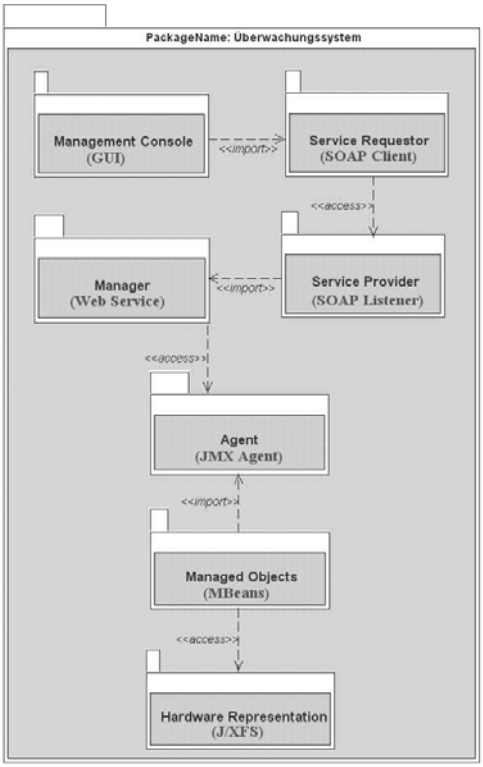

Abbildung 3-21 Prototyp - Top Level Architektur und Realisierung

Eine weitere Verfeinerung dieser Architektur resultiert in:

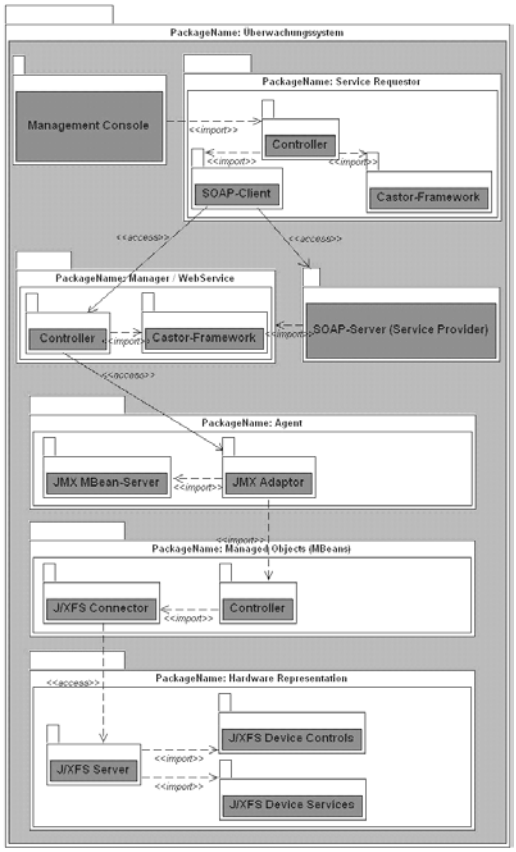

Abbildung 3-22 Prototyp - verfeinerte Top Level Architektur

3.5 Feinentwurf (Detail Design)

Ausgangssituation für den Feinentwurf ist die Grobdefinition der Architektur im Grobentwurf. Ziel des Feinentwurfs ist es, Daten und Prozesse so detailliert darzustellen, dass diese ohne größere Schwierigkeiten direkt implementiert werden können. Die in *Kapitel 3.4 – Grobentwurf (Top Level Design)* vorgestellte Architektur ermöglicht die Realisierung von einfachen Managementsystemen mit beliebigem Funktionsumfang sowie beliebiger Verteilung und Skalierbarkeit. Eine maximal mögliche (sinnvolle) Verteilung wird durch das Deployment-Diagramm in *Abbildung 3.2.3 Prototyp – Deployment bei maximaler Verteilung* dargestellt. Da es sich hier aber um einen Prototypen für ein solches System handelt, sollen folgende Einschränkungen gemacht werden:

- Das zu überwachende Bankperipheriegerät ist ein einfacher Geldautomat.
- Die zu bearbeitenden Anfragen (*siehe Kapitel 3.2.4 – Akteure und Anwendungsfälle*) beschränken sich auf die folgenden:
  - Anfrage über den aktuellen Status des Geldautomaten
  - Anfrage über die aktuellen Kassettenstände des Geldautomaten
- Die vom Geldautomaten ausgehenden Ereignisse (*siehe Kapitel 3.2.4 – Akteure und Anwendungsfälle*) werden als einfache Strings dargestellt.
- Die Verteilung gestaltet sich wie in *Abbildung 3-24 Prototyp – Realisierte Verteilung* dargestellt.

3.5.1 Deployment

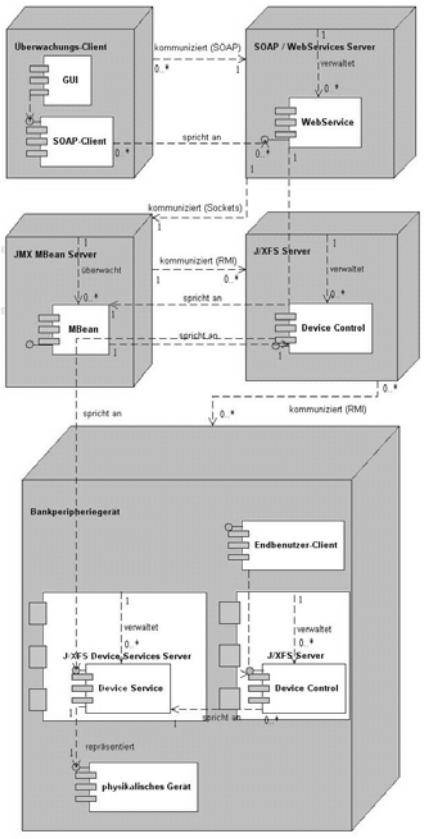

Abbildung 3-23 Prototyp - Deployment bei maximaler Verteilung

Je nach Anzahl der zu überwachenden Peripheriegeräte kann es sinnvoll sein, die Geschäftslogik auf einzelne Netzwerkknoten, d.h. Server, zu verteilen, welche bei besonders hohen Anforderungen selbst noch skaliert werden können. Die Verteilung gestaltet sich in diesem Fall folgendermaßen:

- Überwachungs-Client
  Der Überwachungs-Client befindet sich, außer in der Entwicklungsphase selbstverständlich nicht auf dem Bankperipheriegerät. Es kann sich hierbei um einen beliebigen Client handeln (Java, C#, VB usw.), da dieser sich über das offen gelegte Web Service Interface verbindet. Beim vorliegenden Prototyp sind Clients in Java und C# realisiert.
- SOAP Web Services Server
  Meist ist es sinnvoll, den SOAP Web Services Server, welcher in den meisten Fällen von einem Web Application Server gehostet wird, als zentrale Middleware-Plattform einzusetzen, um weitere Dienste zentral anbieten zu können. Die Kommunikation mit dem Agent erfolgt dann genauso wie die Kommunikation mit anderen Backend-Systemen. Der Management Web Service repräsentiert hierbei den Manager Level.
- JMX MBean Server
  Der JMX MBean Server, welcher den Agent Level repräsentiert, dient der Verwaltung der Ressourcen. Dieser kann als einzelner unabhängiger Knoten realisiert werden, um zum Beispiel Ressourcengruppen bzw. Hierarchien aufzubauen.
- J/XFS Server
  Ein J/XFS Server dient der Verwaltung verschiedener Endgeräte. Auch dieser Server kann als einzelner Knoten realisiert sein.
- Bankperipheriegerät
  Die unterste Ebene des Backends bildet das zu überwachende Peripheriegerät. Dieses dient als Laufzeitumgebung für die einzelnen Device Services, welche jeweils direkt eine Ressource repräsentieren. Weiterhin ist auf diesem Peripheriegerät meist eine grafische Endbenutzerapplikation installiert, wodurch die eigentliche Benutzung des Gerätes erst möglich wird. Hierzu ist es unter Umständen sinnvoll einen J/XFS Server zu installieren, welcher nur zur Überwachung der an dieses Gerät angeschlossenen Ressourcen dient.

Die einfachste Verteilung, welche für die Überwachung einer geringen Gerätemenge geeignet ist, sieht nur zwei Rechner vor: den Rechner des Clients und den Rechner des Endgeräts, auf welchem sich der Rest der Infrastruktur befindet. Der Prototyp ist auf diese Art und Weise realisiert, kann aber ohne Probleme auf das oben vorgestellte Szenario umgestellt werden. Die Verteilung im prototypischen Ansatz wird durch folgende Abbildung verdeutlicht.

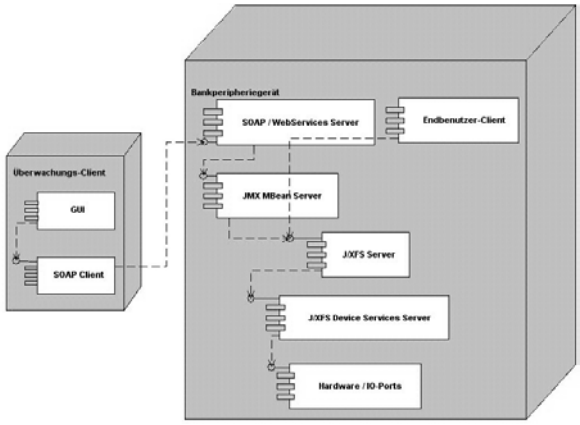

Abbildung 3-24 Prototyp - realisierte Verteilung

3.5.2 Managementdaten

Die auszutauschenden Managementdaten werden, wie bereits angesprochen, als XML Schema dargestellt. Eine Ausnahme hierzu bilden die auf dem Endgerät aufgetretenen Ereignisse, welche der Einfachheit halber als einfache String repräsentiert werden. Die Erstellung der XML Schemata, welche später dem Databinding-Framework (siehe *Kapitel 3.4.3.3 – Castor 0.9.3*) übergeben werden, lassen sich entweder mühsam per Hand oder mit Hilfe grafischer Tools[1] erstellen. Da der Aufbau eines XML Schemas objektorientierten Konzepten folgt, lässt sich dieses problemlos aus bereits bestehenden Objektstrukturen, in diesem Fall J/XFS Objektstrukturen, ableiten. Die folgenden Abbildungen zeigen den groben Aufbau der XML Schema für den Status und die Kassetteninformationen des Geldautomaten.

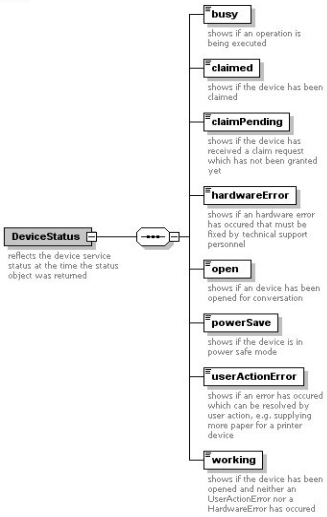

Abbildung 3-25 Prototyp - DeviceStatus XML Schema

[1] Der Autor verwendet hierzu das Programm ´XML Spy 4´. Weitere Informationen unter http://www.xmlspy.com.

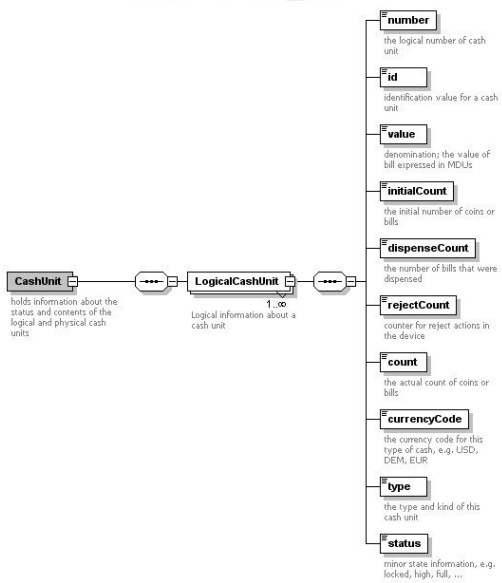

Abbildung 3-26 Prototyp - CashUnit XML Schema

3.5.3 Informationsfluss

Leider existiert in der UML kein Diagramm, womit sich Informationsflüsse sinnvoll darstellen lassen. Doch gerade bei komplexen Systemen, in welchen die Informationen verschiedene Schichten durchlaufen und dabei jeweils ein schichtspezifisches Format aufweisen, ist solch ein Diagramm sinnvoll. Die auf den nächsten Seiten folgenden (nicht standardisierten) Diagramme stellen diesen Informationsfluss dar. Die erste Abbildung zeigt den Informationsfluss eines Ereignisses vom Endgerät, die zweite den eines Kommandos vom Client.

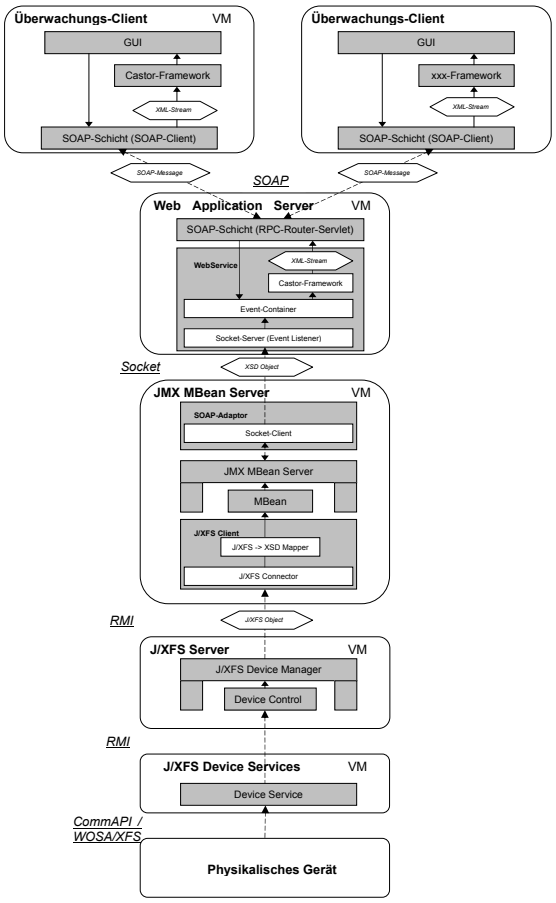

Abbildung 3-27 Prototyp - Informationsflussdiagramm - Ereignis

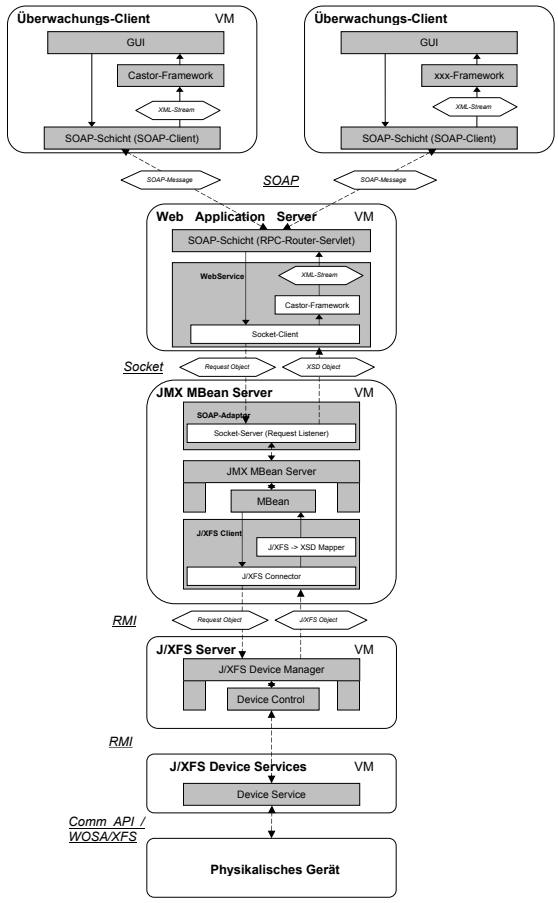

Abbildung 3-28 Prototyp - Informationsflussdiagramm - Kommando

Die abgerundeten Rechtecke stellen hierbei einzelne Prozesse bzw. VMs (virtuelle Maschinen) dar, welche miteinander kommunizieren, um die Anfragen, Ereignisse und Managementdaten zwischen Benutzer des Systems und Endgerät durchzureichen. Die verwendeten Kommunikationstechnologien sind: SOAP, Sockets, RMI[1] und die Comm API[2] bzw. alternativ WOSA/XFS. Innerhalb der Prozesse bzw. VMs werden der Aufbau und die zur Kommunikation benötigten Komponenten dargestellt. Die aktuelle Form des Datums ist als Sechseck dargestellt: Es handelt sich hierbei je nach Schicht um ein J/XFS Objekt, eine XML Schema Instanz, eine SOAP-Nachricht oder applikationsspezifische Objekte.

3.5.4 Detaillierte Statik und Dynamik

Mit diesem Vorwissen lassen sich nun leicht die zu erstellenden Klassen bestimmen. Es handelt sich hierbei um Klassen für:

- die GUI des Überwachungspersonals
- die GUI des Endbenutzers (nur zu Demonstrationszwecken)
- die Managementdaten (durch Databinding Framework automatisch generiert)
- den SOAP-Client
- den Web Service
- den JMX Adaptor
- den JMX Agenten
- die MBeans
- den J/XFS Connector
- die J/XFS Logik
- den Mapper von J/XFS nach XML Schema

Die restlichen Strukturen und Funktionalitäten *(siehe Kapitel 3.3 – Konzeptionelles Modell)* werden durch die eingesetzten Produkte, Technologien und APIs erbracht. Es macht daher, nach Meinung des Autors, wenig Sinn, umfassende Klassen- oder Sequenzdiagramme zu erstellen, da diese enorme Ausmaße besitzen würden.

[1] RMI (Remote Method Invocation) ist das verteilte Komponentenmodell der Java-Plattform.

[2] Java API zum Zugriff auf serielle und parallele Schnittstellen.

3.6 Implementierung

Auf die Implementierung soll an dieser Stelle nicht näher eingegangen werden, da es in dieser Arbeit vorrangig um die Darstellung der verwendeten Konzepte geht. Die Implementierung als Umsetzung dieser Konzepte ist, aufgrund der bereits vorhandenen Frameworks, nicht komplex. Dennoch ist der Aufwand vergleichsweise hoch, da die meisten Frameworks sich noch in einem sehr frühen Stadium befinden und zurzeit kaum nennenswerte Unterstützung seitens der Literatur existiert.

KAPITEL **4**

Bewertung

Das ist eben nichts bewunderungswürdiges,
man darf nur die rechten Tasten zu rechter Zeit treffen,
so spielt das Instrument selbst.
(Johann Sebastian Bach)

4 Bewertung

Das Ziel dieser Arbeit war es, in die Gebiete der Management- und Web Services-Architekturen einzuführen, deren Vor- und Nachteile, Einsatzgebiete und Entwicklung zu diskutieren und sie, sowie entsprechende ´state of the art´-Lösungen, dem Leser näher bringen. Des Weiteren sollten, basierend auf diesen Erkenntnissen, durch die prototypische Konzeption und Realisierung eines Überwachungssystems für Bankperipheriegeräte die enormen Möglichkeiten und Potentiale aufzeigt werden, wie durch die Kombination von Management- und Web Services-Architekturen lose-gekoppelte, plattformunabhängige, leicht integrier- und adaptierbare Management-Systeme erstellt werden können. Der erste Aspekt der Zielstellung wird durch *Kapitel 2 – Architekturen, Technologien und Paradigmen* abgedeckt, in dem eine detaillierte Betrachtung dieser Bereiche erfolgt. Der zweite Aspekt, welcher den praktischen Teil dieser Arbeit bildet, ist in *Kapitel 3 – Ein Überwachungssystem für Bankperipheriegeräte* vorgestellt.

Während der Beschäftigung mit der gesamten Thematik, erkannte der Autor eine Reihe von Entwicklungen und Trends, welche im Folgenden kurz vorgestellt werden sollen:

- Verteilte Systeme
  Die Entwicklungen der letzten Jahre haben gezeigt, dass verteilte Systeme sich durch eine bessere Anpassbarkeit an physikalische, organisatorische und softwaretechnische Anforderungen auszeichnen. Daher werden monolithische Systeme und auf einzelnen PCs installierte Software verschwinden und durch verteilte Systeme ersetzt werden.
- Einheitliche Benutzeroberflächen
  Mit der HTML-Oberfläche des Browsers steht erstmals in der Geschichte eine weltweit fast einheitliche und akzeptierte Benutzeroberfläche zur Verfügung. So ist es sinnvoll diese auch in anderen Bereichen als dem Internet zu verwenden, um unnötige Einarbeitungszeit einzusparen und damit Kosten zu senken. Diesen Trend kann man in allen fast allen Bereichen, angefangen von einfachen Administrationsoberflächen bis hin zu Managementsystemen beobachten.

  Insbesondere wird diese Möglichkeit durch neue Ansätze wie JavaServer Faces[1] und Windows Forms[2] eine weitere Bedeutung erlangen.
- Integrierte Managementsysteme
  Die Arbeit mit Managementsystemen aller Arten hat gezeigt, dass es am sinnvollsten ist, ein Managementsystem zu verwenden, mit dem im Idealfall alle Bereiche des Managements, d.h. Enterprise-, Anwendungs-, Informations-, System- und Netzwerkmanagement (siehe *Kapitel 2.2.1.1 – Der Begriff ´Management´*) abgedeckt werden können. Weiterhin sollen bereits bestehende Systeme und zugekaufte Bausteine problemlos in diese Lösung integriert werden können und eine einheitliche Managementoberfläche existieren.
- Anwendungsmanagement
  Dieser Teilbereich des Managements wird in Zukunft enorm an Bedeutung gewinnen. Insbesondere durch die steigende Anzahl und Komplexität verteilter Systeme im Internet-Bereich, dem Ausbau des B2B und der erwarteten Explosion des B2C durch mobile Endgeräte, wird das Management der unternehmenskritischen Anwendungen immer wichtiger. Hierzu zählt insbesondere auch das Management der Infrastruktur, die durch die Dienste verwendet wird, d.h. zum Beispiel das Management der J2EE-Applikationsserver, in welchem JMX vermutlich seine Hauptanwendung finden wird.
- Kundenausrichtung
  Früher interessierte es den Kunden kaum, wie bzw. mit Hilfe welcher Technologien ein System implementiert wurde. Entscheidend waren anderen Kriterien wie zum Beispiel Funktionalität und Zuverlässigkeit. Heute, da das Marketing auch in diesen Bereich Einzug gehalten hat, stellt sich die Situation anders dar. Wenn der Kunde ´.NET´ oder ´EJBs´ will, muss man dies erfüllen, ob man es für sinnvoll hält oder nicht. Die strikte Trennung von Analyse- und Entwurfsphase, in der man sich erst in der Entwurfsphase auf die Technologien und Programmiersprachen festlegt, hat spätestens jetzt (leider) nur noch rein akademischen Charakter.

[1] Hierbei handelt es sich um ein auf dem JavaServer Pages Standard aufbauendes Framework, womit sich auf einfache Art und Weise HTML Client GUIs erstellen lassen. Es gleicht eine Art Java Swing für HTML Browser.

[2] Eine kurze Beschreibung dieser Technologie findet sich in *Kapitel 2.1.6.3 – Microsoft .NET.*

- Open-Source Software
  In Unternehmen wird zunehmend Open-Source Software eingesetzt, da gegenüber proprietären Lösungen eine Reihe von Vorteilen, wie Herstellerunabhängigkeit, hohe Flexibilität und Wartung in ´Internet-Zeit´ bestehen und die früheren Probleme bezüglich mangelnder Skalierbarkeit und Benutzerfreundlichkeit weitgehend an kommerzielle Produkte angepasst sind.
- ´just in time integration´ und ´ hot plug and play partnerships´
  Ein Wunsch, der durch das Web Service Paradigma erfüllt werden könnte, ist die unkomplizierte Integration der Geschäftsprozesse verschiedener Unternehmen in ein Prozessnetzwerk. Diese neue Form des Unternehmens wird als ´virtuelles Unternehmen´ bezeichnet, welches sehr schnell und dynamisch auf Änderungen reagieren kann, um seine Effizienz zu erhöhen. Hierzu gehört zum Beispiel die dynamische Integration neuer Partner zu jedem beliebigen Zeitpunkt, d.h. das dynamische Hinzufügen, Austauschen und Entfernen von Diensten und Partnern.
- High-Level Programmierung
  Der Anwendungsentwickler wird durch Infrastrukturdienste zur Verfügung stellende Container, durch transparente und austauschbare Kommunikations-infrastrukturen usw. immer mehr von der Low-Level Programmierung entlastet und kann sich auf seine eigentliche Aufgabe, die Entwicklung der Geschäftslogik, konzentrieren. Das Berufsfeld des Anwendungsentwicklers wird sich daher dahingehend ändern, dass in den meisten Bereichen immer mehr das betriebswirtschaftliche Wissen von Interesse ist und sich die Beschäftigung mit komplexen Algorithmen und Datenstrukturen auf den Security-Bereich beschränken wird.
- Code-Generierung
  Ein weiterer Trend, den viele Entwickler argwöhnisch beobachten, ist die automatische Code-Generierung. Gerade im Bereich Web Services oder XML, in dem bei wirklichen Geschäftsanwendungen seitenlange Deployment-Deskriptoren oder WSDL-Dokumente entstehen, wäre es wirtschaftlich nicht vertretbar diese manuell zu erzeugen. Andere Bereiche, für die dieser Fall zutrifft, sind die Generierung von Proxies und der gesamte Bereich des Databinding (siehe *Kapitel 3.4.3.3 - Castor 0.9.3*).
- Interoperabilität
  Gewünscht sind nicht-proprietäre Systeme, welche möglichst unabhängig sind,

um sich neuen Anforderungen effizient anpassen zu können. Die Unabhängigkeit bezieht sich hierbei auf Aspekte der Plattform, der Programmiersprache, der Anwendung, des Transportprotokolls und natürlich des Herstellers. Insbesondere mit der Verbreitung der Web Services Technologie wird die Diskussion um gute oder schlechte Programmiersprachen allmählich der Vergangenheit angehören. In Zukunft wird es wichtiger sein, seine Schnittstellen sauber und klar zu definieren.

Über den Reifegrad der vorgestellten neuen Architekturen, d.h. Web Services und JMX, lässt sich folgendes sagen: Zurzeit halten sich viele konservative Unternehmen mit der Entwicklung und dem Einsatz von Web Service Architekturen noch zurück. Dies hat drei Gründe: Erstens wird auf gut funktionierende und getestete Entwicklungsumgebungen für Web Services gewartet. Zweitens bestehen Bedenken gegenüber dem Einsatz von Technologien, deren Fundament sich noch im Entwicklungs- und Standardisierungsprozess befindet. Das dritte ernstzunehmende Problem ist die zurzeit noch bestehende Transaktionsproblematik dieser Technologien. Insgesamt sind Frameworks wie Microsoft .NET oder Sun ONE heute sicherlich noch nicht reif, um darauf geschäftskritische Anwendungen abzustützen. Ein pragmatisches und dennoch risikoarmes Vorgehen besteht darin, mit Hilfe von vorgeschalteten Web Services bereits existierende Funktionalität nach außen hin verfügbar zu machen und dadurch Mehrwert zu schaffen. JMX liegt zurzeit in Version 1.0 vor und insbesondere im Distributed Services Level (Manager Level) sind noch nicht alle Komponenten spezifiziert. Daher ist auch hier mit zum Teil weitreichenden Änderungen und Ergänzungen zu rechnen, so dass diese Technologie für die Entwicklung umfassender Managementlösungen zunächst nur prototypisch verwendet werden sollte. JMX wird, zumindest zunächst, eine Hauptrolle im Management von Java-Anwendungen und hier insbesondere im Management von J2EE Applikationsservern spielen. Sun sieht vor, JMX in die nächste J2EE Version als Managementplattform für die J2EE-Plattform zu integrieren.

Dem Autor sind weiterhin, insbesondere im praktischen Teil der Arbeit, folgende Probleme und Unzulänglichkeiten der verwendeten Technologien und Produkte aufgefallen, die hier in Bezug auf künftige Weiterentwicklungen des Prototyps angesprochen werden sollen:

- Apache SOAP 2.2 ist (außer über bestimmte Workarounds) nicht mit Microsoft SOAP for Visual Studio .NET interoperabel.

- Apache Axis 1.0 alpha 2 ist zurzeit nicht in der Lage Client-Proxies zu generieren.
- Der Umstieg von Apache SOAP auf Apache Axis erfordert das Neuerstellen der Clients, da die API sich stark geändert hat.
- MS SOAP for Visual Studio .NET hat bei der Anbindung an Web Services, welche nicht auf MS SOAP basieren, Probleme mit allen Datentypen, welche sowohl über einfache Typen als auch Felder von einfachen Typen hinausgehen. An diesen Stellen muss auf Low-Level Programmierung zurückgegriffen werden.
- Visual Studio .NET liefert bei der Web Service Client-Proxy Generierung keine Fehlermeldungen, wenn die WSDL unbekannte Typen enthält. Der generierte Proxy ist in diesem Fall allerdings funktionsuntüchtig.

Nach Einarbeitung in die in dieser Arbeit vorgestellten Architekturen einschließlich der Betrachtung der praktischen Umsetzung, d.h. des Prototyps, müsste es einer für den IT-Bereich qualifizierten Person bzw. einem Team möglich sein, sowohl Management- als auch Web Services-Architekturen zu analysieren, zu konzipieren und zu realisieren und die vorgestellten Vorteile der Kombination dieser in eigenen Ansätzen zu berücksichtigen. Damit wurde das für diese Arbeit gestellte Ziel (siehe *Kapitel 1.2 – Zielsetzung*) erreicht.

KAPITEL **5**

Zusammenfassung

Glendower – I can call spirits from the vasty deep.
Hotspur – Why, so can I, or so can any man;
But will they come when you do
call for them?
(William Shakespeare, King Henry IV, Part 1, Act 3, Scene 1)

5 Zusammenfassung

In diesem Kapitel sollen die wichtigsten Punkte dieser Arbeit noch einmal kurz dargestellt werden. Es werden die Vor- und Nachteile der vorgestellten Technologien angeschnitten und deren Auswirkungen auf die unterschiedlichsten Bereiche skizziert. Weiterhin soll ein kurzer Ausblick dazu dienen, die Entwicklungsrichtung darzustellen und Ideen für weitere Beschäftigung mit dieser Thematik zu liefern.

Wie auch der Rest der IT-Technologien entwickelt sich das Internet permanent und für die Allgemeinheit unbemerkt weiter. Es begann als ´document web´ welches ausschließlich zum Austausch von meist wissenschaftlichen Dokumenten diente und daher nur von Militär und Forschung genutzt wurde. Später entdeckten die Unternehmen den Wert des Internets – zunächst entstand eine Reihe von Web-Seiten, auf denen die Unternehmen sich und ihre Produkte präsentierten, später entstand aufgrund der enormen Resonanz der Wunsch, mit den Kunden direkt auf diesem Weg Transaktionen abzuwickeln. So entwickelte sich das ´application web´ mit der Möglichkeit, Geschäfte und auch Bezahlvorgänge direkt auf dieser Plattform abzuwickeln. Nun besteht der Bedarf Unternehmensprozesse über das Internet mit anderen Unternehmen zu integrieren, was zum so genannten ´service web´ führt. Dies wird durch Web Services ermöglicht, welche in *Kapitel 2.1 - Web Services Architekturen* vorgestellt wurden. Es besteht eine Reihe von Technologien, die diesen Schritt schon seit einiger Zeit hätten wahr werden lassen können, aber folgende Probleme haben dies bisher verhindert:

- Es existiert keine vernünftige Integration der zurzeit aufkommenden Embedded Devices und mobilen Endgeräten in bestehende Infrastrukturen.
- Traditionelle IT-Systeme sind zu unflexibel, d.h. bestehende Beziehungen zwischen Unternehmen müssen mühsam aufgebaut und gepflegt werden, so dass zum Beispiel spontane Lieferantenwechsel nicht möglich sind.
- Kontextsensitivität der Dienste ist nur in den seltensten Fällen zu finden.
- Es besteht eine Reihe von Interoperabilitätsproblemen.
- Die bestehenden Lösungen sind komplex, teuer und proprietär.
- Es existiert eine Reihe von Problemen mit Firewalls.

Web Services lösen diese Probleme durch einen anderen Ansatz: Es werden bereits bestehende, seit Jahren im Internet erfolgreich eingesetzte, Technologien als Basis verwendet und eine zusätzliche Schicht an Funktionalität aufgesetzt, so dass es sich nicht um einen völlig neuen Ansatz, sondern nur eine Erweiterung handelt. Die Web Service Vision soll hierbei in Phasen verwirklicht werden, so dass jeder Baustein genug Zeit hat zu reifen und sich selbständig zu entwickeln. Zurzeit befinden wir uns in der ersten Phase, die vollständige Vision soll ca. 2005 verwirklicht sein. Die Zahl der SOAP-Implementierungen nimmt stetig zu, so dass mittlerweile für fast alle Programmiersprachen und Plattformen eine SOAP-Anbindung vorhanden und damit wichtige Schritte in die richtige Richtung geschehen sind.

Der Vollständigkeit halber soll an dieser Stelle das grundlegende Web Service Modell noch einmal dargestellt werden, welches die Grundlage der gesamten Web Service Vision darstellt:

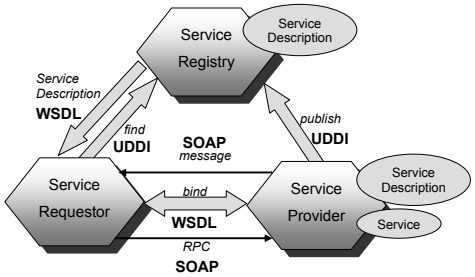

Abbildung 5-1 Web Service Modell

Die drei Instanzen der Web Service Architektur werden durch Sechsecke dargestellt und die Pfeile repräsentieren die Interaktionen. Neben den Interaktionspfeilen sind die dafür zuständigen Protokolle bzw. Technologien angeführt. Der Service Provider stellt verschiedene Dienste auf seinem Server zur Verfügung und veröffentlicht die Web Service Beschreibung in einer globalen Registry, wo diese von jedem aufgefunden werden kann. Der Service Requestor sucht in dieser Registry nach für ihn geeigneten bzw. benötigten Diensten. Als Antwort auf seine Suchanfrage erhält er die Schnittstellenbeschreibung des gewünschten Dienstes und ist nun in der Lage, auf diesen zuzugreifen, um ihn zum Beispiel in eigene Applikationen einzubinden.

Web Services werden sich voraussichtlich mehr oder weniger in allen IT-Bereichen wieder finden:

- Intranet
  Im Intranet-Bereich werden Web Services wahrscheinlich eine eher untergeordnete Rolle spielen, da die dort existierenden Protokolle, zumindest heute noch, die größeren Vorteile besitzen.
- Internet, Kundenanbindung
  Die Anbindung von Kunden auf verschiedenen Vertriebswegen und Kanälen über das Internet ist sicher das Paradebeispiel für die Verwendung der Web Service Technologie.
- EAI (Enterprise Application Integration)
  Web Services werden hierbei intern zur vereinfachten Anbindung der Legacy-Systeme verwendet. Zurzeit werden sie aufgrund der noch fehlenden Transaktionsverwaltung als ´EAI-lite´ bezeichnet.
- Value-Chains, Supply-Chains
  Der zweite enorm wichtige Bereich ist die Unternehmenskommunikation mit der einfachen Möglichkeit des Aufbaus von Supply-Chains und Value-Chains. Hierzu gehören auch die so genannten ´hot plug and play partnerships´, d.h. die dynamische Integration neuer Partner zu jedem beliebigen Zeitpunkt.
- Interoperabilität zwischen J2EE und MS-Architekturen
  Da Web Services Anwendungen über das Internet verfügbar machen, aber nicht der Implementierung dieser Anwendungen dienen, verkörpern sie keine Konkurrenz zu den weit verbreiteten J2EE- und MS-Architekturen. Sie können aber eine problemlose Interoperabilität zwischen diesen gewährleisten, so dass die Implementierung des benötigten Dienstes transparent ist. So ist es das erste Mal in der Geschichte ohne Probleme möglich, unabhängig von den zugrunde liegenden Technologien und Architekturen zu kommunizieren.

Den prophezeiten Mehrwert werden Web Services aber erst erbringen, wenn die Vision und damit die entsprechenden Protokolle vollständig sind, gut funktionierende und akzeptierte Entwicklungsumgebungen existieren und die insbesondere von Sun und Microsoft propagierten ´Smart Web Services´ Verbreitung gefunden haben. Sowohl Entwickler als auch IT-Manager sollten sich spätestens jetzt mit der Web Services Technologie auseinander setzen, entsprechende Seminare besuchen und

im Idealfall erste Feldversuche, zum Beispiel in Form von vergebenen Diplomarbeiten, unternehmen.

Die sinnvolle Nutzung der neu entstehenden, verteilten Systeme erfordert bei gleichzeitig wachsender Komplexität aber ein adäquates Management für heterogene Netz- und Systemumgebungen, das die Sicherstellung eines effektiven und effizienten Einsatzes eines Rechnernetzes und eines verteilten Systems ermöglichen muss. In der globalisierten, arbeitsteiligen Wirtschaft sind vernetzte und verteilte IT-Systeme unabdingbare Infrastrukturen. Netze, Systeme und Anwendungen werden immer komplexer und eine manuelle Konfiguration und Überwachung sind aufgrund der Komplexität nicht mehr möglich. Das sichere Funktionieren der Systeme und die schnelle Anpassbarkeit an geänderte Anforderungen bekommen zunehmend unternehmenskritische Bedeutung. Es müssen daher geeignete Managementsysteme zur Administration und Überwachung dieser IT-Infrastruktur eingesetzt werden. Unter Management werden je nach Kontext verschiedene Bereiche verstanden, im Allgemeinen klassifiziert man Management aber in der folgenden Art und Weise:

- Enterprise-Management – das Management eines gesamten Unternehmens
- Anwendungs-Management – das Management unterschiedlicher Anwendungen
- Informations-Management – das Management von Datenbeständen
- System-Management – das Management von umfassenden Rechnernetzen
- Netzwerk-Management – das Management von Netzdiensten und –komponenten

Bei der Betrachtung des Themengebiets ´Management´ in Bezug auf seine technologische Basis existieren drei grundsätzliche Entwicklungstendenzen, welche in dieser Arbeit detailliert vorgestellt wurden:

- OSI-Management
- SNMP-Management
- Management der ´dritten Generation´ (z.B. JMX)

Auch an dieser Stelle soll der Vollständigkeit halber das grundlegende Management-Architektur Modell noch einmal dargestellt werden, das die Basis aller Management-Lösungen darstellt:

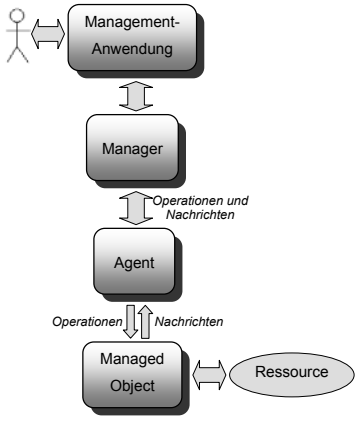

Abbildung 5-2 Management-Architektur Modell

Die verwalteten logischen und physikalischen Ressourcen werden durch Managed Objects repräsentiert, welche eine Abstraktion auf die für das Management relevanten Aspekte der Komponente darstellen und sich aus Attributen, Operationen, Nachrichten und Verhalten zusammensetzen. Die Management-Anwendung stellt das Frontend zum menschlichen Part des Management, also einem Administrator oder Operator, dar. Dabei handelt es sich meist um eine mehr oder weniger komplexe grafische Benutzeroberfläche. Der Manager nimmt Anfrage und Aufträge der Management-Anwendung entgegen und leitet diese an den Agenten weiter. Auf der anderen Seite erhält er vom Agenten sowohl Antworten auf diese Operationen als auch Meldungen, welche er an die Management-Anwendung weiterleitet. Der Agent verwaltet die Managed Objects und stellt das Bindeglied zwischen diesen und dem Manager dar.

Wie auch das Internet befinden sich Managementsysteme und –architekturen in einem stetigen Wandel. Heutige Netze und insbesondere die auf diesen Netzen operierenden verteilten Anwendungen unterliegen einem immer rascheren Wandel, der sich auch im Bedarf an ein flexibles, skalierbares, dynamisches, offenes,

standardisiertes und eben den neuen Anforderungen angepasstes Management äußert. Die klassischen Managementarchitekturen sind diesen Anforderungen kaum mehr gewachsen. Es existieren daher eine Reihe neuerer Ansätze, welche als ´Management der dritten Generation´ bezeichnet werden. Nach Meinung des Autors wird auch in diese Systeme früher oder später die Web Services Technologie Einzug halten. Zunächst wird das Management der Web Services und der dahinter liegenden Funktionalität an Bedeutung gewinnen und später zunehmend auch die Offenlegung der Managementfunktionalität als Web Service. Der in dieser Arbeit vorgestellte Prototyp für ein modernes Überwachungssystem für Bankperipheriegeräte zeigt unter anderen, wie der zweite Punkt auf einfache Art und Weise realisiert werden kann. Weiterhin wird gezeigt, wie sich die aktuellen Technologien in den Bereichen Management, Web Services und Bankperipheriezugriffsarchitekturen kombinieren lassen, um ein flexibles, skalierbares, dynamisches, offenes und auf Standards basierendes Überwachungssystem für Bankperipheriegeräte zu realisieren.

VERZEICHNISSE **V**

Abkürzungen, Abbildungen, Tabellen

Abkürzungsverzeichnis

API	Application Programming Interface
ACSE	Association Control Service Element
ADO	ActiveX Data Objects
ARM	Application Response Measurement
ASF	Apache Software Foundation
ASN.1	Abstract Syntax Notation One
ASP	Application Service Provider
ASP	Active Server Pages
B2B	Business-to-Business
B2C	Business-to-Customer
BLOB	Binary Large Object
CDATA	Character Data
CERN	Conseil Européen pour la Recherche Nucléaire
CIM	Common Information Model
CLR	Common Language Runtime
CMIP	Common Management Information Protocol
CMISE	Common Management Information Service Element
CMOT	Common Management Information Service over TCP/IP
CORBA	Common Object Request Broker Architecture
CRMS	Customer Relationship Management System
DBMS	Database Management System
DBVS	Datenbankverwaltungssystem
DCOM	Distributed Component Object Model
DMTF	Distributed Management Task Force

DNA	Distributed InterNet Architecture
DNS	Domain Name Service
DoD	Department of Defense
DOM	Document Object Model
DTD	Document Type Definition
EAI	Enterprise Application Integration
EAR	Enterprise Archive
EBNF	Erweiterte Backus Naur Form
ebXML	Electronic Business XML
EDI	Electronic Data Interchange
EIS	Enterprise Information System
EJB	Enterprise Java Bean
ERPS	Enterprise Resource Planning System
FCAPS	Fault-, Configuration-, Accounting-, Performance-, Security-Management
FDI	Financial Device Interface
FTP	File Transfer Protocol
GUI	Graphical User Interface
HEMS	High-Level Entity Management System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPR	Hypertext Transfer Protocol Reliable
HTTPS	HTTP über SSL
IAB	Internet Architecture Board
IDE	Integrated Development Environment
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
INS	Internation Network Services
IP	Internet Protocol
ISO	International Standardisation Organisation
J/XFS	Extensions for Financial Services for the Java Platform
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JavaPOS	Java Point of Sales
JAX/RPC	Java API for XML Remote Procedure Calls
JAXB	Java API for XML Binding
JAXM	Java API for XML Messaging
JAXP	Java API for XML Processing
JAXR	Java API for XML Registries
JCA	Java Connector Architecture
JCP	Java Community Process
JDBC	Java Database Connectivity
JDMK	Java Dynamic Management Kit
JIDM	Joint Inter-Domain Management
JMAPI	Java Management API
JMS	Java Message Service
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface
JRE	Java Runtime Environment
JSP	JavaServer Pages
JSR	Java Specification Request
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
MIB	Management Information Base
MIM	Management Information Model
MIT	Massachusetts Institute of Technology
MO	Managed Object
MOB	Managed Object Boundary
MOC	Managed Object Class
MOM	Message Oriented Middleware
MQ	Message Queuing
MSB	Manage-ability Service Broker
MSIL	Microsoft Intermediate Language
MVC	Model View Controller
NC	Network Client
OASIS	Organization for the Advancement of Structured Information Standards
ODBC	Open Database Connectivity
OO	Objektorientierung
OODBMS	Object Oriented Database Management System
ORB	Object Request Broker
OSI	Open Systems Interconnect
P2P	Point-to-Point
PCDATA	Parsable Character Data
PDA	Personal Digital Assistant
PDF	Portable Document Format
PI	Processing Instruction
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RMI-IIOP	Remote Method Invocation over Internet Inter-ORB Protocol
ROI	Return of Investment
ROSE	Remote Operations Service Element
RPC	Remote Procedure Call
RTF	Rich Text Format
SAML	Security Assertion Markup Language
SAX	Simple API for XML
SB	Selbstbedienung
SCMS	Supply Chain Management System
SGML	Standard Gerneralized Markup Language
SGMP	Simple Gateway Monitoring Protocol
SMASE	Systems Management Application Service Element
SMI	Structure of Management Information
SMS	Short Message Service
SMTP	Simple Message Transfer Protocol
SOAP	Simple Object Access Protocol
SOAP-DSIG	Simple Object Access Protocol – Digital Signatures
SOD	Service on Demand
SPI	Service Provider Interface
SQL	Standard Query Language
SSL	Secure Socket Layer

Sun ONE	Sun Open Network Environment
TCP	Transmission Control Protocol
TMN	Telecommunication Management Network
TPM	Transaction Processing Monitor
TTM	Time to market
UDDI	Universal Description, Discovery, Integration
UDP	User Datagram Protocl
UEP	Überwachungspersonal
UML	Unified Modelling Language
UPP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VB	Visual Basic
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WAR	Web Archive
WBEM	Web-based Enterprise Management
WML	Wireless Markup Language
WOSA	Windows Open Services Architecture
WOSA/MAPI	Windows Open Services Architecture for Messaging
WOSA/SNA	Windows Open Services Architecture for Systems Network Architecture
WOSA/XFS	Windows Open Services Architecture / Extensions for Financial Services
WOSA/XRT	Windows Open Services Architecture for Realtime Market Data
WSCM	Web Services Component Model
WSDL	Web Service Description Language
WSFL	Web Services Flow Language
WSIL	Web Services Inspection Language
WWW	World Wide Web
XHTML	XML Hypertext Markup Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSL FO	Extensible Stylesheet Language Formatting Objects
XSLT	Extensible Stylesheet Language Transformations

Abbildungsverzeichnis

Abbildung 2-1 Entwicklungsgeschichte der Web Services 17
Abbildung 2-2 Web Service Perspektiven 18
Abbildung 2-3 ASP, SOD und Web Services (Quelle: [CBDIWS], S.11) 19
Abbildung 2-4 E-Commerce Szenario der Zukunft 20
Abbildung 2-5 Web Services Modell 22
Abbildung 2-6 Web Services Evolution (Quelle: [WSRHRN],S.5) 24
Abbildung 2-7 Web Services Stack (Quelle: [WSCA], S.10) 26
Abbildung 2-8 Beziehung Meta-Sprache, Markup-Sprache, Dokument 32
Abbildung 2-9 Einfaches XML-Dokument 33
Abbildung 2-10 Einfache DTD 34
Abbildung 2-11 Programmierschnittstellen und Parser 35
Abbildung 2-12 DOM-Baum 36
Abbildung 2-13 Einfaches Namensraum-Beispiel 38
Abbildung 2-14 Einfaches XML Schema 40
Abbildung 2-15 Benutzung der im Schema deklarierten Elemente 40
Abbildung 2-16 Prinzip der XSLT 41
Abbildung 2-17 Prinzip der XSL FO 43
Abbildung 2-18 Aufbau einer SOAP Nachricht (Quelle: [SKSM00], S.81) 45
Abbildung 2-19 Beispiel für einen SOAP-Request 46
Abbildung 2-20 Beispiel für eine SOAP-Response 46
Abbildung 2-21 SOAP Kommunikation 47
Abbildung 2-22 WSDL Entitäten und deren Beziehungen 50
Abbildung 2-23 WSDL Dokument 51
Abbildung 2-24 UDDI Datenstruktur 54
Abbildung 2-25 Web Services Modell und Technologien 54
Abbildung 2-26 Sun ONE Basisarchitektur (Quelle: [JM9/01A]) 59
Abbildung 2-27 Sun ONE Smart Web Services Architektur (Quelle: [JM9/01A]) 60
Abbildung 2-28 Sun ONE Architektur Standards (Quelle: [JM9/01A]) 61
Abbildung 2-29 .NET Architektur (Quelle: [MSNET], S.13) 63
Abbildung 2-30 Evolution der Wertschöpfungskette 66
Abbildung 2-31 Klassifikation des Managements 71
Abbildung 2-32 Dimensionen des Managements (Quelle: [SJM01], S.8) 73
Abbildung 2-33 INS Umfrage 1999 (Quelle: [SJM01],S.7) 75
Abbildung 2-34 Historie verschiedener Management-Technologien 78
Abbildung 2-35 Teilmodelle einer Management-Architektur 81
Abbildung 2-36 Management-Architektur 82
Abbildung 2-37 Beziehung zwischen Ressource und Managed Object 83
Abbildung 2-38 Funktionale Gliederung des Managements 85
Abbildung 2-39 ISO/OSI-Basisreferenzmodell (Quelle: [ZWSG94],S.14) 87
Abbildung 2-40 OSI-Management Standards 88
Abbildung 2-41 Organisationsmodell des OSI-Managements 89
Abbildung 2-42 OSI-Domänenmodell 90
Abbildung 2-43 ISO Registration-Tree 92
Abbildung 2-44 Beispiel für die MOC Definition 93
Abbildung 2-45 OSI-Dienstelemente in Schicht 7 (Quelle: [STAL93],S. 214) 94
Abbildung 2-46 OSI Funktionsbereiche (FCAPS) 95
Abbildung 2-47 TCP/IP-Modell 96
Abbildung 2-48 Beispiel für die Definition einer SNMP-Variable 97
Abbildung 2-49 Registrierung der definierten SNMP-Variable 98
Abbildung 2-50 Einordnung SNMP in das TCP/IP-Modell 98
Abbildung 2-51 SNMP Kommunikationsmodell (Quelle: [STAL93], S.75) 100
Abbildung 2-52 JMX-Architektur (Quelle: [JMX2], S.21) 105
Abbildung 2-53 Beispiel einer Standard MBean Implementierung 109
Abbildung 2-54 Aufbau des JMX Agenten (Quelle: [JMX2], S.102) 111
Abbildung 2-55 Integrierte Managementarchitektur (Quelle:[HHAS93],S.301) 115
Abbildung 2-56 Windows Open Services Architecture (Quelle: [SNIWOSA]) ... 118
Abbildung 2-57 J/XFS Unabhängigkeiten (Quelle: [JXSO], S.7) 122
Abbildung 2-58 J/XFS Architektur 123
Abbildung 2-59 J/XFS Zugriff (Quelle: [JXBA], S.7) 126
Abbildung 2-60 Ablauf einer J/XFS Operation 127
Abbildung 2-61 J/XFS Benutzungsbeispiel 128
Abbildung 3-1 Offenlegung einer Managementschnittstelle als Web Service. 135
Abbildung 3-2 Überwachung durch Web Services 135
Abbildung 3-3 Überwachung von Web Services 136
Abbildung 3-4 Prototyp – Systemübersicht 138
Abbildung 3-5 Prototyp - Use Case Diagramm 140
Abbildung 3-6 Prototyp – Use Case Nr.1 Aktivitätsdiagramm 143

Abbildung 3-7 Prototyp – Use Case Nr.2 Aktivitätsdiagramm 146
Abbildung 3-8 Prototyp – Use Case Nr.3 Aktivitätsdiagramm 149
Abbildung 3-9 Prototyp – Klassendiagramm 151
Abbildung 3-10 Prototyp - Sequenzdiagramm "Anfrage bearbeiten" 159
Abbildung 3-11 Prototyp - Sequenzdiagramm "Registrierung für Ereignisse" .. 160
Abbildung 3-12 Prototyp - Sequenzdiagramm "Ereignis weiterleiten" 160
Abbildung 3-13 Prototyp - Geräteauswahl-Maske 162
Abbildung 3-14 Prototyp - Kommando-Maske 162
Abbildung 3-15 Prototyp - Ereignis-Maske 163
Abbildung 3-16 Prototyp - Top Level Architektur 164
Abbildung 3-17 Apache SOAP Framework 168
Abbildung 3-18 Einfacher Web Service 169
Abbildung 3-19 Einfacher Apache SOAP Client 169
Abbildung 3-20 XML Data Binding 171
Abbildung 3-21 Prototyp - Top Level Architektur und Realisierung 173
Abbildung 3-22 Prototyp - verfeinerte Top Level Architektur 174
Abbildung 3-23 Prototyp - Deployment bei maximaler Verteilung 176
Abbildung 3-24 Prototyp - realisierte Verteilung 178
Abbildung 3-25 Prototyp - DeviceStatus XML Schema 179
Abbildung 3-26 Prototyp - CashUnit XML Schema 180
Abbildung 3-27 Prototyp - Informationsflussdiagramm - Ereignis 181
Abbildung 3-28 Prototyp - Informationsflussdiagramm - Kommando 182
Abbildung 5-1 Web Service Modell 193
Abbildung 5-2 Management-Architektur Modell 196

Tabellenverzeichnis

Tabelle 2-1 Web Services Standards 25
Tabelle 2-2 Historie des Managements im OSI-Bereich 77
Tabelle 2-3 Historie des Managements im SNMP-Bereich 77
Tabelle 2-4 Neuere Ansätze im Management-Bereich 77
Tabelle 2-5 Unterschiede OSI- und Internet-Management 102
Tabelle 3-1 Prototyp – Use Case Nr.1 Spezifikation 141
Tabelle 3-2 Prototyp – Use Case Nr.1 Ablaufbeschreibung 142
Tabelle 3-3 Prototyp – Use Case Nr.2 Spezifikation 144
Tabelle 3-4 Prototyp – Use Case Nr.2 Ablaufbeschreibung 145
Tabelle 3-5 Prototyp – Use Case Nr.3 Spezifikation 147
Tabelle 3-6 Prototyp – Use Case Nr.3 Ablaufbeschreibung 148
Tabelle 3-7 Prototyp – Fachlexikon 158

Literaturverzeichnis

Bücher:

[HHAS93] Hegering, H.-G., Abeck, S. (1993):
Integriertes Netz- und Systemmanagement.
1. Aufl. 1993, Addison Wesley Verlag

[MERZ99] Merz, Michael (1999):
Electronic Commerce – Marktmodelle, Anwendungen und Technologien.
1. Aufl. 1999, Heidelberg, dpunkt Verlag

[ROSE91] Rose, Marshall T. (1991):
The Simple Book.
1. Aufl. 1991, Prentice-Hall, Inc.

[SKSM00] Scribner, Kennard, Stiver, Mark (2000):
Understanding SOAP.
1. Aufl. 2000, SAMS USA

[STAL93] Stalling, William (1993):
SNMP, SNMPv2, and CMIP – The Practical Guide to Network-Management Standards.
1. Aufl. 1993, Addison-Wesley Publishing Company

Fachzeitschriften:

[CT6/01] Willers, Michael (2001):
Microsoft .NET.
In: c´t, 6/2001, S.253ff

[CUTR01] Cutlip, Rob (2001):
Web Services: The New Web Paradigm.
In: DB2 magazine, 4/2001, S.32ff

[CW8/01a] Bauer, Michael (2001):
Web-Services – Methoden und Tools für verteiltes Computing.
In: Computerwoche extra, 8/2001, S.4ff

[CW8/01b] Erbrich, Thomas (2001):
Microsofts .NET: Werkstatt für Web-Services.
In: Computerwoche extra, 8/2001, S.12ff

[HDMP96] Heiliger, Dieter (1996):
Management-Plattformen.
Seminararbeit
TH Darmstadt

[HMM96] Hoffmann, Mario (1996):
Eine Einführung in das Netzwerk- und Systemmanagement – Schwerpunkt: Modellierung.
Seminararbeit
TH Darmstadt

[iX6/01a] Klever, Nik (2001):
Elementarteilchen – XML-Schema: objektorientierte Dokumenttyp-definitionen.
In: iX, 6/2001, S.62ff

[iX6/01b] Wegelin, Michael (2001):
.NET baut auf XML.
In: iX, 6/2001, S.74ff

[iX6/01c] Wegelin, Michael (2001):
.NET baut auf XML – Die Kommunikationsinfrastruktur der .NET-Server-Familie.
In: iX, 6/2001, S.67ff

[JM3/00a] Großwendt, Volkmar (2000):
Ausgezeichnet – XML – Eine Einführung.
In: Java Magazin, 1/2000, S.79ff

[JM3/00b] Großwendt, Volkmar (2000):
Reine Formsache – Elementstruktur von XML-Dokumenten definieren.
In: Java Magazin, 3/2000, S.81ff

[JM7/00] Großwendt, Volkmar (2000):
Saubere Sache – Vereinfachte Kommunikation im Internet mit SOAP.
In: Java Magazin, 7/2000, S.96ff

[JM12/00] Mathy, Frank (2000):
SOAP Opera – Verteilte Dienste mit SOAP unter Java.
In: Java Magazin, 12/2000, S.20ff

[JM1/01] Schönberger, Dirk (2001):
Im Blickpunkt: Microsoft .NET und die neue Sprache C#.
In: Java Magazin, 1/2001, S.51ff

[JM4/01] Röwekamp, Lars (2000):
Sonniges XML – Wie unterstützt Sun Microsystems das Traumpaar Java/XML?.
In: Java Magazin, 4/2001, S.21ff

[JM5/01a] Holubek, Andreas (2001):
Objekte aufbewahren – Persistenz zum Nulltarif: Kurz- und Langzeitpersistenz mit Java.
In: Java Magazin, 5/2001, S.16ff

[JM5/01b] Knuth, Michael (2001):
Serviceorientiert – Web Service Definition Language – was ist das?.
In: Java Magazin, 5/2001, S.91ff

[JM5/01c] Seemann, Michael (2001):
Fisch sucht Fahrrad? – Mit UDDI fit für B2B?.
In: Java Magazin, 5/2001, S.95ff

[JM7/01] Seemann, Michael (2001):
Seife im Tipi – Die Apache SOAP-Implementierung am Beispiel vorgestellt.
In: Java Magazin, 7/2001, S.97ff

[JM9/01a] Sarakatsanis, Athanasios (2001):
Java-Sicht auf Web Services: Suns Open Net Environment (ONE).
In: Java Magazin, 9/2001, S.40ff

[JM9/01b] Knuth, Michael (2001):
Web Services – die Middleware der Zukunft?.
In: Java Magazin, 9/2001, S.32ff

[JM9/01c] Holubek, Andreas (2001):
Das Web Services Pack als Plattform für die webzentrierte Entwicklung.
In: Java Magazin, 9/2001, S.37ff

[JM11/01] Sarakatsanis, Athanasios (2001):
Sun ONE vs. .NET – die Frameworks im Vergleich.
In: Java Magazin, 11/2001, S.60ff

[JR10/99] Deadman, Richard (1999):
XML as a distributed application protocol.
In: Java Report, 10/1999, S.16ff

[JS1/00] Hranitzky, Norbert (2000):
XML- Der Stein der Weisen?.
In: Java Spektrum, 1/2000, S.49ff

[JS6/00] Studer, Bruno (2000):
Gut verbunden – Die Telekommunikationstechnologien JAIN und JMX.
In: Java Spektrum, 6/2000, S.43ff

[JS1/01] Starke, Gernot (2001):
Überwachung einplanen – Management Java-basierter Client/Server Anwendungen.
In: Java Spektrum, 1/2001, S.47ff

[JS2/01] Frotscher, Thilo (2001):
Kommunikation mit dem Simple Object Access Protocol.
In: Java Spektrum, 2/2001, S.60ff

[JS4/01] Wieland, Thomas (2001):
Schwarz auf Weiss – Beschreibung von WebServices.
In: Java Spektrum, 4/2001, S.22ff

[JXAPG] J/XFS Forum (2000):
J/XFS Application Programming Guide Version 1.0.
J/XFS Forum

[JXBA] J/XFS Forum (2000):
J/XFS Base Architecture – Programmer´s Reference Version 1.1.
J/XFS Forum

[JXCO] J/XFS Forum (2000):
J/XFS Conceptional Overview.
J/XFS Forum

[JXSO] J/XFS Forum (2000):
J/XFS System Overview Version 1.0.
J/XFS Forum

[KREG01] Kreger, H. (2001):
Java Management Extensions for application management.
In: IBM Systems Journal, VOL 40, NO 1, 2001

[KTOSI00] Kiesling, Tobias (2000):
OSI-Management.
Seminararbeit
Universität München

[MTJMX01] Miklitz, Thomas (2001):
Java Management Extensions.
Seminararbeit
TU Darmstadt

[OBS6/00a] Vasters, Clemens (2000):
SOAP – die Lösung aller Interoperabilitätsprobleme?.
In: OBJEKTspektrum, 6/2000, S.26ff

[OBS6/00b] Willers, Michael (2000):
Microsoft .NET – das programmierbare Web.
In: OBJEKTspektrum, 6/2000, S.19ff

[OBS6/00c] Stahl, Michael (2000):
Microsoft .NET – Evolution oder Revolution?.
In: OBJEKTspektrum, 6/2000, S.14ff

[OBS2/01] Heckert, Erik (2001):
Bankensoftware im Wandel.
In: OBJEKTspektrum, 2/2001, S.18ff

[RHJK99] Reiser, Helmut (1999):
Einsatz des Java Dynamic Management Kit für flexible, dynamische Managementsysteme.
In: PIK – Praxis der Informationsverarbeitung und Kommunikation, 4/1999, Saur Verlag

[SJM01] Schönwälder, Jürgen (2001):
Netzwerkmanagement.
Vorlesungsskript
TU Braunschweig

[SNIWOSA] Siemens Nixdorf Retail and Banking Systems GmbH (1999):
WOSA.
Internes Dokument

[STM98] Stiel, Hadi (1998):
Mühsamer Weg zum umfassenden Management.
In: Funkschau 23/98, S.70ff

[VSM00] Saca, Valentin (2000):
Neue Organisationsmodelle: Verteilte und erweiterbare Managementsysteme.
Seminararbeit
Universität München

[WNPV00] Wincor Nixdorf GmbH & Co. KG (2000):
ProView Version 2.2.
Internes Dokument

[ZREJB01] Zacharias, Roger (2001):
EJB-Applikationsserver.
Seminararbeit
FH Gießen-Friedberg

[ZRXML01] Zacharias, Roger (2001):
XML – Das ´Esperanto´ des E-Commerce?.
Seminararbeit
FH Gießen-Friedberg

[ZWSG94] Zorn, W., Schreiner, G. (1994):
Netzwerkmanagement.
Seminararbeiten
Universität Karlsruhe

www-Dokumente:

[APACHE] Apache XML Project
www-Dokument vom 21.05.2001:
http://xml.apache.org

[BRAUJ01] Brauer, Jürgen:
Die IT-Branche – Beschreibungssprachen HTML, XML ...
www-Dokument vom 04.04.2001:
http://home.t-online.de/home/juergen.brauer/seite03a.htm

[CBDIWS] CBDI Forum
Guide to Web Services.
www-Dokument vom 19.07.2001:
http://www.cbdiforum.com

[DRV01] Dragan, Richard V.:
Will Sun be the ONE for Web Services?.
www-Dokument vom 26.11.2001:
http://www.zdnet.com

[FMLJ01] Fleury, Marc, Lindfors, Juha:
Enabling Component Architectures with JMX.
www-Dokument vom 23.07.2001:
http://www.onjava.com/pub/a/onjava/2001/02/01/jmx.html

[GSMM01] Gambhir, Sahil, Muchmore, Michael W.:
Web Services: Revolution in the making.
www-Dokument vom 26.11.2001:
http://www.zdnet.com

[HJWS01] Hart, James:
How Web Services came to be – Tracing the ancestry of Web Services.
www-Dokument vom 29.11.2001:
http://www.webservicesarchitect.com/content/articles/hart01.asp

[JAVAW1] Kadel, Rich:
Components for the rest of us.
www-Dokument vom 10.08.2000:
http://www.javaworld.com/javaone98/j1-98-ejb.html

[JAVAW2] Goff, Max:
Java in the management sphere.
www-Dokument vom 20.11.2001:
http://www.javaworld.com/javaworld/jw-10-20001

[JAVAW3] Bowker, Todd:
Superior app management with JMX.
www-Dokument vom 23.10.2001:
http://www.javaworld.com/javaworld/jw-06-2001

[JAVAW4] Borck, James R.:
Leaders of the Web services pack.
www-Dokument vom 02.10.2001:
http://www.javaworld.com/javaworld/jw-09-2001

[JAVAW5] Rommel, John:
Will Web services jump-start the software slump?.
www-Dokument vom 07.09.2001:
http://www.javaworld.com/javaworld/jw-08-2001

[JAVAW6] Bansal, Sonal, Pal, Gaurav:
The Web at your (machine´s) service.
www-Dokument vom 10.10.2001:
http://www.javaworld.com/javaworld/jw-09-2001

[JAVAW7] Sriniva, Raghavan N.:
Web services hits the Java scene, Part 1.
www-Dokument vom 02.10.2001:
http://www.javaworld.com/javaworld/jw-09-2001

[JBLUE] Sun Microsystems Inc.:
J2EE Blueprints.
www-Dokument vom 02.08.2001:
http://www.javasoft.com/j2ee/blueprints

[JMX1] Sun Microsystems, Inc.:
Java Management Extensions White Paper.
www-Dokument vom 02.10.2001:
http://java.sun.com/jmx

[JMX2] Sun Microsystems, Inc. (2000):
Java Management Extensions Instrumentation and Agent Specification, v1.0.
www-Dokument vom 02.10.2001:
http://java.sun.com/jmx

[JMX3] Sun Microsystems, Inc.:
Java Dynamic Management Kit White Paper.
www-Dokument vom 02.10.2001:
http://java.sun.com/jmx

[JMX4] Sun Microsystems, Inc.:
Getting Started with the Java Dynamic Management Kit 4.2.
www-Dokument vom 02.10.2001:
http://java.sun.com/jmx

[JMX5] Sun Microsystems, Inc.:
Java Dynamic Management Kit 4.2 Tutorial.
www-Dokument vom 02.10.2001:
http://java.sun.com/jmx

[MMP01] McMahon, Peter:
Web Services – A Guide for Business and IT Managers.
www-Dokument vom 29.11.2001:
http://www.dotnet.za.net

[MOT01] Modi, Tarak:
JMX Makes App Management Simple.
www-Dokument vom 19.11.2001:
http://www.devx.com

[MSNETD] Microsoft
.NET Defined.
www-Dokument vom 23.05.2001:
http://www.microsoft.com/net/

[MSNETI] Microsoft
An Introduction to Microsoft .NET.
www-Dokument vom 27.05.2001:
http://www.microsoft.com/net/intro.asp

[NEWB01] Newport, Billy:
JMX – Network Management Software.
www-Dokument vom 23.07.2001:
http://www.theserverside.com/resources/articles/jmx/article.html

[OFM01] Sun Microsystems Inc. – Open Finance Magazine:
Driving Force.
www-Dokument vom 08.11.2001:
http://www.sun.com/finance/openfin/spring01

[ONE] Sun Microsystems, Inc.
Sun Open Net Environment (SUN ONE).
www-Dokument vom 23.05.2001:
http://www.sun.com/software/sunone/

[ONESA] Sun Microsystems, Inc.:
Sun Open Network Environment (Sun ONE) Software Architecture.
www-Dokument vom 03.11.2001:
http://www.sun.com/sunone/

[PHIPS01] Phipps, Simon
Building the Swarm.
www-Dokument vom 25.05.2001:
http://java.sun.com/features/2001/04/swarm.html

[RAWS01] Ryman, Arthur
Understanding Web Services.
www-Dokument vom 17.11.2001:
http://www-106.ibm.com/developerworks/webservices

[SADAF01] Savarese, Daniel F.:
ONE Web to Bind Them.
www-Dokument vom 01.11.2001:
http://www.devx.com

[SAPT01] SAP Deutschland
mySAP Technologie.
www-Dokument vom 23.05.2001:
http://www.sap-ag.de/germany/technology/index.asp

[SEEM01] Seemann, Michael
Spürhunde im Web.
www-Dokument vom 25.05.2001:
http://entwickler.com/jm/ausgaben/2001/6/artikel/8/online.shtml

[SHINKA] Shinka Technologies:
Shinka Integration Server 1.2 Whitepaper.
www-Dokument vom 09.10.2001:
http://www.shinka.de/pdf/whitepaper.pdf

[SM01a] Stahl, Michael
Paradigm Shift for Agility through Web Services.
www-Dokument vom 19.12.2001:
http://www.stahl.de

[SM01b] Stahl, Michael
Workshop on Web Services - SE Day 2001.
www-Dokument vom 19.12.2001:
http://www.stahl.de

[SM01c] Stahl, Michael
Stets zu Diensten: Web-Services im Überblick.
www-Dokument vom 19.12.2001:
http://www.stahl.de

[SOAP] W3C:
Simple Object Access Protocol (SOAP) 1.1.
www-Dokument vom 21.06.01
http://www.w3.org/TR/SOAP

[VAVE01] Vasudevan, Venu:
A Web Services Primer.
www-Dokument vom 07.08.2001:
http://www.xml.com/lpt/a/2001/04/04/webservices

[VCRE01] Vawter, Chad, Roman, Ed:
J2EE vs. Microsoft .NET.
www-Dokument vom 21.12.2001:
http://www.theserverside.com/resources

[VEB01] van Eyle, Ben
Web Services – A Business Perspective on Platform Choice.
www-Dokument vom 25.12.2001:
http://www.theserverside.com/resources/articles/WebServices

[VITIRIS] Killdara Corporation
Vitiris Web Services Platform.
www-Dokument vom 19.12.2001:
http://www.killdara.com

[WIC01] Wiggers, Chanoch:
Sun and Web Services – The Competition Heats up.
www-Dokument vom 08.09.2001:
http://www.webservicesarchitect.com/content/articles/wiggers.asp

[WNPCE] Wincor Nixdorf GmbH & Co. KG:
ProClassic Enterprise.
www-Dokument vom 02.01.2002:
http://www.myproclassic.com

[WSCA] IBM Software Group:
Web Services Conceptual Architecture (WSCA 1.0).
www-Dokument vom 09.10.2001:
http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[WSGART] Gartner Research Note COM-13-5985:
How Web Services Mean Business.
www-Dokument vom 12.11.2001:
http://www.gartner.com

[WSRHRN] CBDI Forum
Web Services – Right Here, Right Now.
www-Dokument vom 19.07.2001:
http://www.cbdiforum.com

ANHANG **A**

XML Schemata, Quelltext

A1 XML Schemata für die Managementdaten

DeviceStatus.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
 attributeFormDefault="unqualified">
    <xs:element name="DeviceStatus">
        <xs:annotation>
            <xs:documentation>reflects the device service status at the time the status object
            was returned</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="busy" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>shows if an operation is being
                        executed</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="claimed" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>shows if the device has been
                        claimed</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="claimPending" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>shows if the device has received a claim
                        request which has not been granted yet</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="hardwareError" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>shows if an hardware error has occured
                        that must be fixed by technical support
                        personnel</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="open" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>shows if an device has been opened for
                        conversation</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="powerSave" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>shows if the device is in power safe
                        mode</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="userActionError" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>shows if an error has occured which can be resolved
                        by user action, e.g. supplying more paper for a printer device
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="working" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>shows if the device has been opened and
                        neither an UserActionError nor a HardwareError has
                        occured</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

CashUnit.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
 attributeFormDefault="unqualified">
    <xs:element name="CashUnit">
        <xs:annotation>
            <xs:documentation>holds information about the status and contents of the
            logical and physical cash units</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="LogicalCashUnit" maxOccurs="unbounded">
                    <xs:annotation>
                        <xs:documentation>Logical information about a cash unit
                        </xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="number" type="xs:int">
                                <xs:annotation>
                                    <xs:documentation>the logical number of cash unit
                                    </xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="id" type="xs:string">
                                <xs:annotation>
                                    <xs:documentation>identification value for a cash unit
                                    </xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="value" type="xs:int">
                                <xs:annotation>
                                    <xs:documentation>denomination; the value of bill expressed
                                    in MDUs</xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="initialCount" type="xs:int">
                                <xs:annotation>
                                    <xs:documentation>the initial number of coins or bills
                                    </xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="dispenseCount" type="xs:int">
                                <xs:annotation>
                                    <xs:documentation>the number of bills that were
                                    dispensed</xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="rejectCount" type="xs:int">
                                <xs:annotation>
                                    <xs:documentation>counter for reject actions in the
                                    device</xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="count" type="xs:int">
                                <xs:annotation>
                                    <xs:documentation>the actual count of coins or bills
                                    </xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="currencyCode" type="xs:string">
                                <xs:annotation>
                                    <xs:documentation>the currency code for this type of cash,
                                    e.g. USD, DEM, EUR</xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="type" type="xs:string">
                                <xs:annotation>
                                    <xs:documentation>the type and kind of this cash unit
                                    </xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="status" type="xs:string">
                                <xs:annotation>
                                    <xs:documentation>minor state information, e.g. locked, high,
                                    full, ...</xs:documentation>
                                </xs:annotation>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

A2 Quellcode

GUI des Überwachungspersonals

```
package com.wincornixdorf.pce.rzdiplomarbeit.gui.operatorclientgui;

// connected client
import com.wincornixdorf.pce.rzdiplomarbeit.soap.operatorclient.RzProtoOperatorClient;

// J2SE 1.4
import javax.swing.UIManager;
import java.awt.*;
import java.util.Observer;
import java.util.Observable;
import java.util.Vector;


/**
 * GUI for RzProtoOperatorClient - MAIN entry point. <br>
 * half-automatically created (with GUI-designer). <br>
 * observer of client to get events from the client.
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoOperatorGui implements Observer {

	boolean packFrame = false;
	// the connected client (soap = /, axis=2)
	private RzProtoOperatorClient soapClient;
	// the main GUI frame
	private RzProtoOperatorFrame frame;


	/**
	 * this method is automatically called by the connected client,
	 * if an event came in. <br>
	 * put event on GUI.
	 *
	 * @param   o Observable
	 * @param   arg event object
	 */
	public void update(Observable o, Object arg) {
		frame.eventOccurred((Vector) arg);
	}

	/**
	 * returns a reference to the connected client. <br>
	 * TODO: Interface für Client schreiben, welches z.B. Methode sendCommand() hat,
	 * um GUI unabhängig vom Client zu machen!
	 *
	 * @return    reference to connected client
	 */
	public RzProtoOperatorClient getClient() {
		return this.soapClient;
	}


	/**
	 * The class constructor. <br>
	 * Construct the graphical user interface.
	 *
	 */
	public RzProtoOperatorGui() {
		// initiating RzProtoOperatorClient
		soapClient = new RzProtoOperatorClient();
		// registering as observer to get events from client
		soapClient.addObserver(this);
		// main frame
		frame = new RzProtoOperatorFrame(this);
		//Validate frames that have preset sizes
		//Pack frames that have useful preferred size info, e.g. from their layout
		if (packFrame) {
			frame.pack();
		}
		else {
			frame.validate();
		}
		//Center the window
		Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
		Dimension frameSize = frame.getSize();
		if (frameSize.height > screenSize.height) {
			frameSize.height = screenSize.height;
		}
		if (frameSize.width > screenSize.width) {
			frameSize.width = screenSize.width;
		}
		frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height –
		frameSize.height) / 2);
		frame.setVisible(true);
	}

	/**
	 * main method. <br>
	 * starts the GUI.
	 *
	 * @param   args command line arguments
	 *
	 */
	public static void main(String[] args) {
		try {
			UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
		}
		catch(Exception e) {
			e.printStackTrace();
		}
		RzProtoOperatorGui gui = new RzProtoOperatorGui();
	}
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.gui.operatorclientgui;

// J2SE 1.4
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import java.io.*;
import java.util.Vector;


/**
 * The main frame for the Operator GUI.
 *
 * @author   Roger Zacharias
 * @version  1.0
 *
 */
public class RzProtoOperatorFrame extends JFrame {

	// the parent gui of this frame
	private RzProtoOperatorGui parent = null;

	/**
	 * called if "choose device" button is pressed. <br>
	 * send "is device there?" request.
	 */
	void chooseDeviceButton_actionPerformed(ActionEvent e) {
		// write to status bar
		String deviceName = deviceNameField.getText() + "@" + hostNameField.getText();
		this.statusBar.setText("Zugriff auf Gerät <" + deviceName + ">");
		// check if specified device exists
		String  result = null;
		boolean exists;
		try {
			exists = this.parent.getClient().isDeviceExisting(deviceName);
		}
		catch (Exception ex) {
			JOptionPane.showMessageDialog(this, ex.getMessage(), "Fehler bei
			Geräteauswahl", JOptionPane.ERROR_MESSAGE);
			ex.printStackTrace();
			return;
		}
		if (!exists) {
			JOptionPane.showMessageDialog(this, "Das Gerät <" + deviceName + "> existiert
			nicht!", "Fehler bei Geräteauswahl", JOptionPane.ERROR_MESSAGE);
		}
	}

	/**
	 * called if "send command" button pressed. <br>
	 * send command request.
	 */
	void sendCommandButton_actionPerformed(ActionEvent e) {
		// write to status bar
		this.statusBar.setText("Sende Kommando an Gerät <" + deviceNameField.getText() +
				"@" + hostNameField.getText() + ">");
		String deviceName = deviceNameField.getText() + "@" + hostNameField.getText();


		// check which command is selected
		if (this.buttonGroup1.isSelected(this.cassetteInfoButton.getModel())) {
			String result = null;
			try {
				result = this.parent.getClient().sendCassetteCountersRequest(deviceName);
			}
			catch (Exception ex) {
				JOptionPane.showMessageDialog(this, ex.getMessage(), "Fehler bei
				Kommando", JOptionPane.ERROR_MESSAGE);
				ex.printStackTrace();
				return;
			}
			// insert text at top
			commandResultArea.insert(result + "\n", 0);
			// scroll back to top
			this.commandResultArea.setCaretPosition(0);
			// write to status bar
			this.statusBar.setText("Kassetteninformationen eingetroffen!");
		}
		else if (this.buttonGroup1.isSelected(this.deviceStateButton.getModel())) {
			String result = null;
			try {
				result = this.parent.getClient().sendStatusRequest(deviceName);
			}
			catch (Exception ex) {
				JOptionPane.showMessageDialog(this, ex.getMessage(), "Fehler bei
				Kommando", JOptionPane.ERROR_MESSAGE);
				ex.printStackTrace();
				return;
			}
			// insert text at top
			commandResultArea.insert(result + "\n", 0);
			// scroll back to top
			this.commandResultArea.setCaretPosition(0);
			// write to status bar
			this.statusBar.setText("Statusinformationen eingetroffen!");
		}
	}

	/**
	 * called if "register for events" checkBox activated/deactivated. <br>
	 * send register request.
	 */
	void registerForEventsCheckBox_actionPerformed(ActionEvent e) {
		// check if register request or unregister request
		if (this.registerForEventsCheckBox.isSelected()) {
			// write to status bar
			this.statusBar.setText("Registriere diesen Client für Ereignisse von Gerät <" +
			deviceNameField.getText() + "@" + hostNameField.getText() + ">");
			String result = null;
			try {
				result = this.parent.getClient().sendRegisterForNotificationsRequest();
			}
			catch (Exception ex) {
				JOptionPane.showMessageDialog(this, ex.getMessage(), "Fehler bei
				Kommando", JOptionPane.ERROR_MESSAGE);
				ex.printStackTrace();
				return;
			}
			this.eventHistoryArea.insert(result + "\n", 0);
		}
		else {
			// write to status bar
```

```
                this.statusBar.setText("Deregistriere diesen Client für Ereignisse von Gerät <" +
                deviceNameField.getText() + "@" + hostNameField.getText() + ">");
                String result = null;
                try {
                        result = this.parent.getClient().sendDeregisterForNotificationsRequest();
                }
                catch (Exception ex) {
                        JOptionPane.showMessageDialog(this, ex.getMessage(), "Fehler bei
                        Kommando",
                        JOptionPane.ERROR_MESSAGE);
                        ex.printStackTrace();
                        return;
                }
                this.eventHistoryArea.insert(result + "\n", 0);
        }
}

/**
 * automatically called by client if an event occurred. <br>
 * put events on GUI.
 *
 * @param   events Vector of new events
 */
public void eventOccurred(Vector events) {
        // write to status bar
        this.statusBar.setText("Ereignis(se) eingetroffen!");
        for(int i=0;i<events.size();++) {
                this.eventHistoryArea.insert(events.elementAt(i).toString() + "\n", 0);
        }
}


JPanel contentPane;
JMenuBar jMenuBar1 = new JMenuBar();
JMenu jMenuFile = new JMenu();
JMenuItem jMenuFileExit = new JMenuItem();
JMenu jMenuHelp = new JMenu();
JMenuItem jMenuHelpAbout = new JMenuItem();
ImageIcon image1;
ImageIcon image2;
ImageIcon image3;
JLabel statusBar = new JLabel();
BorderLayout borderLayout1 = new BorderLayout();
JTabbedPane jTabbedPane1 = new JTabbedPane();
JPanel Kommandos = new JPanel();
JPanel Gerät = new JPanel();
JLabel jLabel1 = new JLabel();
JRadioButton cassetteInfoButton = new JRadioButton();
JRadioButton deviceStateButton = new JRadioButton();
JPanel Ereignisse = new JPanel();
JButton sendCommandButton = new JButton();
JLabel jLabel2 = new JLabel();
JTextArea commandResultArea = new JTextArea();
ButtonGroup buttonGroup1 = new ButtonGroup();
JTextArea eventHistoryArea = new JTextArea();
JLabel jLabel3 = new JLabel();
VerticalFlowLayout verticalFlowLayout2 = new VerticalFlowLayout();
JPanel jPanel1 = new JPanel();
VerticalFlowLayout verticalFlowLayout1 = new VerticalFlowLayout();
VerticalFlowLayout verticalFlowLayout3 = new VerticalFlowLayout();
JLabel jLabel4 = new JLabel();
JPanel jPanel2 = new JPanel();
JButton chooseDeviceButton = new JButton();
JLabel jLabel5 = new JLabel();
JTextField deviceNameField = new JTextField();
JLabel jLabel6 = new JLabel();
JTextField hostNameField = new JTextField();
JPanel jPanel5 = new JPanel();
GridLayout gridLayout1 = new GridLayout();
JPanel jPanel3 = new JPanel();
GridLayout gridLayout2 = new GridLayout();
JLabel jLabel7 = new JLabel();
JPanel jPanel4 = new JPanel();
JScrollPane eventScrollPane = new JScrollPane();
JScrollPane commandScrollPane = new JScrollPane();
JCheckBox registerForEventsCheckBox = new JCheckBox();

/**
 * The class constructor. <br>
 * constructs the frame.
 *
 * @param   parent parent gui
 */
public RzProtoOperatorFrame(RzProtoOperatorGui parent) {
        this.parent = parent;
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
                jbInit();
        }
        catch(Exception e) {
                e.printStackTrace();
        }
}


/**
 * component initialization.
 */
private void jbInit() throws Exception  {
        image1 = new
        ImageIcon(com.wincornixdorf.pce.rzdiplomarbeit.gui.operatorclientgui.
        RzProtoOperatorFrame.class. getResource("openFile.gif"));
        image2 = new ImageIcon(com.wincornixdorf.pce.rzdiplomarbeit.gui.operatorclientgui.
        RzProtoOperatorFrame.class.getResource("closeFile.gif"));
        image3 = new ImageIcon(com.wincornixdorf.pce.rzdiplomarbeit.gui.operatorclientgui.
        RzProtoOperatorFrame.class.getResource("help.gif"));

        //setIconImage(Toolkit.getDefaultToolkit().createImage(RzProtoOperatorFrame.
        class.getResource("[Your Icon]")));
        contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(borderLayout1);
        this.setSize(new Dimension(619, 428));
        this.setTitle("Überwachungssystem");
        statusBar.setText(" ");
        jMenuFile.setText("Datei");
        jMenuFileExit.setText("Beenden");
        jMenuFileExit.addActionListener(new ActionListener()  {
                public void actionPerformed(ActionEvent e) {
                        jMenuFileExit_actionPerformed(e);
                }
        });
        jMenuHelp.setText("Hilfe");
        jMenuHelpAbout.setText("Über");
        jMenuHelpAbout.addActionListener(new ActionListener()  {
                public void actionPerformed(ActionEvent e) {
                        jMenuHelpAbout_actionPerformed(e);
                }
        });
        jTabbedPane1.setTabPlacement(JTabbedPane.LEFT);
        jTabbedPane1.setMinimumSize(new Dimension(600, 200));
        jLabel1.setFont(new java.awt.Font("Serif", 3, 16));
        jLabel1.setText("Kommandoauswahl");
        Kommandos.setLayout(verticalFlowLayout2);
        cassetteInfoButton.setFont(new java.awt.Font("Dialog", 0, 14));
        cassetteInfoButton.setText("Kassetteninformationen");
        deviceStateButton.setFont(new java.awt.Font("Dialog", 0, 14));
        deviceStateButton.setText("Gerätestatus");
        sendCommandButton.setText("Kommando senden!");
        sendCommandButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        sendCommandButton_actionPerformed(e);
                }
        });
        jLabel2.setFont(new java.awt.Font("Serif", 3, 16));
        jLabel2.setText("Kommandoergebnis");
        commandResultArea.setRequestFocusEnabled(false);
        commandResultArea.setToolTipText("");
        commandResultArea.setEditable(false);
        commandResultArea.setText("Hier steht das Ergebnis des Kommandos!");
        eventHistoryArea.setRequestFocusEnabled(false);
        eventHistoryArea.setEditable(false);
        eventHistoryArea.setText("Hier stehen die vom Gerät eingetroffenen Ereignisse");
        jLabel3.setFont(new java.awt.Font("Serif", 3, 16));
        jLabel3.setText("Eingetroffene Ereignisse");
        Ereignisse.setLayout(verticalFlowLayout1);
        Gerät.setLayout(verticalFlowLayout3);
        jLabel4.setFont(new java.awt.Font("Serif", 3, 16));
        jLabel4.setMaximumSize(new Dimension(262, 17));
        jLabel4.setMinimumSize(new Dimension(362, 17));
        jLabel4.setPreferredSize(new Dimension(200, 17));
        jLabel4.setText("Geräteauswahl");
        chooseDeviceButton.setText("Gerät wählen!");
        chooseDeviceButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        chooseDeviceButton_actionPerformed(e);
                }
        });
        jPanel2.setLayout(gridLayout1);
        jLabel5.setFont(new java.awt.Font("Dialog", 0, 14));
        jLabel5.setHorizontalAlignment(SwingConstants.LEFT);
        jLabel5.setHorizontalTextPosition(SwingConstants.LEFT);
        jLabel5.setText("Gerätename");
        deviceNameField.setToolTipText("");
        deviceNameField.setText("Dummy_CashDispenser");
        jLabel6.setFont(new java.awt.Font("Dialog", 0, 14));
        jLabel6.setHorizontalAlignment(SwingConstants.LEFT);
        jLabel6.setHorizontalTextPosition(SwingConstants.LEFT);
        jLabel6.setText("Hostname");
        hostNameField.setText("localhost");
        jPanel3.setLayout(gridLayout2);
        jLabel7.setFont(new java.awt.Font("Serif", 3, 16));
        jLabel7.setText("Ereignisregistrierung");
        contentPane.setMaximumSize(new Dimension(600, 200));
        contentPane.setMinimumSize(new Dimension(600, 200));
        contentPane.setPreferredSize(new Dimension(600, 200));
        Gerät.setPreferredSize(new Dimension(600, 200));
        commandScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_
        SCROLLBAR_ALWAYS);
        commandScrollPane.setAutoscrolls(true);
        commandScrollPane.setPreferredSize(new Dimension(495, 170));
        commandScrollPane.setToolTipText("");
        eventScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_
        SCROLLBAR_ALWAYS);
        eventScrollPane.setPreferredSize(new Dimension(495, 230));
        registerForEventsCheckBox.setFont(new java.awt.Font("Dialog", 0, 14));
        registerForEventsCheckBox.setToolTipText("");
        registerForEventsCheckBox.setText("Für Ereignisse registrieren!");
        registerForEventsCheckBox.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        registerForEventsCheckBox_actionPerformed(e);
                }
        });
        jMenuFile.add(jMenuFileExit);
        jMenuHelp.add(jMenuHelpAbout);
        jMenuBar1.add(jMenuFile);
        jMenuBar1.add(jMenuHelp);
        this.setJMenuBar(jMenuBar1);
        contentPane.add(statusBar, BorderLayout.SOUTH);
        contentPane.add(jTabbedPane1, BorderLayout.CENTER);
        Kommandos.add(jLabel1, null);
        Kommandos.add(deviceStateButton, null);
        Kommandos.add(cassetteInfoButton, null);
        Kommandos.add(jPanel1, null);
        jPanel1.add(sendCommandButton, null);
        Kommandos.add(jLabel2, null);
        Kommandos.add(commandScrollPane, null);
        commandScrollPane.getViewport().add(commandResultArea, null);
        jTabbedPane1.add(Gerät, "Gerät");
        jTabbedPane1.add(Kommandos,  "Kommandos");
        jTabbedPane1.add(Ereignisse,  "Ereignisse");
        Ereignisse.add(jLabel7, null);
        Ereignisse.add(jPanel4, null);
        jPanel4.add(registerForEventsCheckBox, null);
        Ereignisse.add(jLabel3, null);
        Ereignisse.add(eventScrollPane, null);
        eventScrollPane.getViewport().add(eventHistoryArea, null);
        Gerät.add(jLabel4, null);
        Gerät.add(jPanel2, null);
        jPanel2.add(jLabel5, null);
        jPanel2.add(deviceNameField, null);
        Gerät.add(jPanel3, null);
        jPanel3.add(jLabel6, null);
        jPanel3.add(hostNameField, null);
        Gerät.add(jPanel5, null);
        jPanel5.add(chooseDeviceButton, null);
        buttonGroup1.add(deviceStateButton);
        buttonGroup1.add(cassetteInfoButton);
}


/**
 * File | Exit action performed
 *
 * @param   e
 */
public void jMenuFileExit_actionPerformed(ActionEvent e) {
```

```
            this.parent.getClient().shutdown();
            System.exit(0);
        }

        /**
         * Help | About action performed
         *
         * @param   e
         */
        public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
            RzProtoOperatorFrame_AboutBox dlg = new RzProtoOperatorFrame_AboutBox(this);
            Dimension dlgSize = dlg.getPreferredSize();
            Dimension frmSize = getSize();
            Point loc = getLocation();
            dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height –
            dlgSize.height) / 2 + loc.y);
            dlg.setModal(true);
            dlg.show();
        }

        /**
         * Overridden so we can exit when window is closed
         *
         * @param   e
         */
        protected void processWindowEvent(WindowEvent e) {
            super.processWindowEvent(e);
            if (e.getID() == WindowEvent.WINDOW_CLOSING) {
                jMenuFileExit_actionPerformed(null);
            }
        }
}
```

GUI des Endbenutzers (nur zu Demonstrationszwecken)

```
package com.wincornixdorf.pce.rzdiplomarbeit.gui.enduserclientgui;

// connected client
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.enduserclient.RzProtoEndUserClient;

// J2SE 1.4
import javax.swing.UIManager;
import java.awt.*;


/**
 * GUI for RzProtoEndUserGui - MAIN entry point. <br>
 * half-automatically created (with GUI-designer).
 *
 * @author Roger Zacharias
 * @version 1.0
 */
public class RzProtoEndUserGui {

        boolean packFrame = false;
        // the connected client
        private RzProtoEndUserClient client;


        /**
         * returns a reference to the connected client. <br>
         * TODO: Interface für Client schreiben, welches z.B. Methode sendCommand() hat,
         * um GUI unabhängig vom Client zu machen!
         *
         * @return      reference to connected client
         */
        public RzProtoEndUserClient getClient() {
                return client;
        }

        /**
         * The class constructor. <br>
         * Construct the graphical user interface.
         */
        public RzProtoEndUserGui() {
                // initiating RzProtoEndUserClient
                client = new RzProtoEndUserClient();
                // main frame
                RzProtoEndUserFrame frame = new RzProtoEndUserFrame(this);
                //Validate frames that have preset sizes
                //Pack frames that have useful preferred size info, e.g. from their layout
                if (packFrame) {
                        frame.pack();
                }
                else {
                        frame.validate();
                }
                //Center the window
                Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
                Dimension frameSize = frame.getSize();
                if (frameSize.height > screenSize.height) {
                        frameSize.height = screenSize.height;
                }
                if (frameSize.width > screenSize.width) {
```

```
                        frameSize.width = screenSize.width;
                }
                frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height –
                frameSize.height) / 2);
                frame.setVisible(true);
        }

        /**
         * main method. <br>
         * starts the GUI.
         *
         * @param       args command line arguments
         *
         */
        public static void main(String[] args) {
                try {
                        UIManager.setLookAndFeel(UIManager.
                        getSystemLookAndFeelClassName());
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                new RzProtoEndUserGui();
        }
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.gui.enduserclientgui;

// J2SE 1.4
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import java.util.Vector;


/**
 * The main frame for the End-User GUI.
 *
 * @author Roger Zacharias
 * @version 1.0
 */
public class RzProtoEndUserFrame extends JFrame {

        // the parent gui of this frame
        private RzProtoEndUserGui parent;

        /**
         * called if "ok" button is pressed. <br>
         * send dispense & query request.
         */
        void jButton1_actionPerformed(ActionEvent e) {
                this.statusBar.setText("Zahle Geld aus ...");
                String amountStr = this.amountField.getText();
                int    amount    = Integer.valueOf(amountStr).intValue();

                this.parent.getClient().dispenseMoney(amount);
                this.parent.getClient().queryCashUnits();
                Vector cassetteFields = this.parent.getClient().getCassetteFields();

                String cassette1Text = cassetteFields.elementAt(0).toString();
                String cassette2Text = cassetteFields.elementAt(1).toString();
                String cassette3Text = cassetteFields.elementAt(2).toString();
                String cassette4Text = cassetteFields.elementAt(3).toString();

                this.cassetteField1.setText(cassette1Text);
                this.cassetteField2.setText(cassette2Text);
                this.cassetteField3.setText(cassette3Text);
                this.cassetteField4.setText(cassette4Text);

                this.statusBar.setText(amount + " DM ausgezahlt!");

                // check if a cassette is empty (workaround)
                if (cassette1Text.endsWith(" 0")) {
                        this.statusBar.setText("Geldkassette <ui1> leer!");
                }
                if (cassette2Text.endsWith(" 0")) {
                        this.statusBar.setText("Geldkassette <ui2> leer!");
                }
                if (cassette3Text.endsWith(" 0")) {
                        this.statusBar.setText("Geldkassette <ui3> leer!");
                }
                if (cassette4Text.endsWith(" 0")) {
                        this.statusBar.setText("Geldkassette <ui4> leer!");
                }
        }
```

```
        JPanel contentPane;
        JLabel statusBar = new JLabel();
        JLabel jLabel1 = new JLabel();
        JPanel jPanel1 = new JPanel();
        JLabel jLabel2 = new JLabel();
        JLabel jLabel3 = new JLabel();
        JLabel jLabel4 = new JLabel();
        JLabel jLabel5 = new JLabel();
        GridLayout gridLayout1 = new GridLayout();
        JTextField cassetteField1 = new JTextField();
        JTextField cassetteField2 = new JTextField();
        JTextField cassetteField3 = new JTextField();
        JTextField cassetteField4 = new JTextField();
        JLabel jLabel6 = new JLabel();
        JLabel jLabel7 = new JLabel();
        JPanel jPanel2 = new JPanel();
        FlowLayout flowLayout1 = new FlowLayout();
        XYLayout xYLayout1 = new XYLayout();
        JTextField amountField = new JTextField();
        JButton jButton1 = new JButton();

        /**
         * The class constructor. <br>
         * constructs the frame.
         *
         * @param   parent parent gui
         */
        public RzProtoEndUserFrame(RzProtoEndUserGui gui) {
                this.parent = gui;
                enableEvents(AWTEvent.WINDOW_EVENT_MASK);
                try {
                        jbInit();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                this.parent.getClient().queryCashUnits();
                Vector cassetteFields = this.parent.getClient().getCassetteFields();
                this.cassetteField1.setText(cassetteFields.elementAt(0).toString());
                this.cassetteField2.setText(cassetteFields.elementAt(1).toString());
                this.cassetteField3.setText(cassetteFields.elementAt(2).toString());
                this.cassetteField4.setText(cassetteFields.elementAt(3).toString());
        }

        /**
          * component initialization.
          */
        private void jbInit() throws Exception  {
                //setIconImage(Toolkit.getDefaultToolkit().createImage(RzProtoEndUserFrame.
                class.getResource("[Your Icon]")));
                contentPane = (JPanel) this.getContentPane();
                contentPane.setLayout(xYLayout1);
                this.setSize(new Dimension(408, 300));
                this.setTitle("Endbenutzer-Client");
                statusBar.setText(" ");
                jLabel1.setFont(new java.awt.Font("SansSerif", 1, 12));
                jLabel1.setText("Kassettenstatus");
                jLabel2.setText("Fach");
                jLabel3.setText("ID");
                jLabel4.setText("Notenwert");
                jLabel5.setText("Anzahl Scheine");
                jPanel1.setLayout(gridLayout1);
                cassetteField1.setBackground(Color.lightGray);
                cassetteField1.setEditable(false);
                cassetteField2.setBackground(Color.lightGray);
                cassetteField2.setEditable(false);
                cassetteField3.setBackground(Color.lightGray);
                cassetteField3.setEditable(false);
                cassetteField4.setBackground(Color.lightGray);
                cassetteField4.setEditable(false);
                jLabel6.setFont(new java.awt.Font("Serif", 1, 12));
                jLabel6.setText("Auszahlung");
                jLabel7.setText("Betrag in DM:");
                jPanel2.setLayout(flowLayout1);

                amountField.setColumns(3);
                jButton1.setText("OK");
                jButton1.addActionListener(new java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                jButton1_actionPerformed(e);
                        }
                });
                contentPane.add(jPanel1,  new XYConstraints(5, 27, 390, -1));
                jPanel1.add(jLabel2, null);
                jPanel1.add(jLabel3, null);
                jPanel1.add(jLabel4, null);
                jPanel1.add(jLabel5, null);
                contentPane.add(cassetteField1,  new XYConstraints(5, 49, 390, -1));
                contentPane.add(cassetteField2,  new XYConstraints(5, 75, 390, -1));
                contentPane.add(cassetteField3,  new XYConstraints(5, 101, 390, -1));
                contentPane.add(cassetteField4,  new XYConstraints(5, 127, 390, -1));
                contentPane.add(jLabel6,  new XYConstraints(5, 175, 390, -1));
                contentPane.add(jPanel2,  new XYConstraints(5, 197, 390, -1));
                jPanel2.add(jLabel7, null);
                jPanel2.add(amountField, null);
                jPanel2.add(jButton1, null);
                contentPane.add(jLabel1,  new XYConstraints(5, 5, 390, -1));
                contentPane.add(statusBar,     new XYConstraints(4, 249, 390, -1));
        }

        /**
         * overridden so we can exit when window is closed.
         */
        protected void processWindowEvent(WindowEvent e) {
                super.processWindowEvent(e);
                if (e.getID() == WindowEvent.WINDOW_CLOSING) {
                        System.exit(0);
                }
        }
}
```

SOAP-Client (für Apache SOAP und Apache Axis)

```
package com.wincornixdorf.pce.rzdiplomarbeit.soap.operatorclient;

// castor framework classes for unmarshalling/marshalling from/to xml
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.cashunitschema.*;
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.devicestatusschema.*;

// J2SE 1.4
import java.util.Vector;
import java.net.URL;
import java.net.MalformedURLException;
import java.io.StringWriter;
import java.io.StringReader;
import java.util.Observable;
// Castor 0.9.3
import org.exolab.castor.xml.ValidationException;
import org.exolab.castor.xml.MarshalException;
// SOAP 2.2
import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Response;
import org.apache.soap.rpc.Parameter;
import org.apache.soap.Fault;
import org.apache.soap.Constants;
import org.apache.soap.SOAPException;
// Xerces 1.4.3
import org.w3c.dom.Element;
import org.w3c.dom.Document;
import org.apache.soap.util.xml.DOM2Writer;


/**
 * SOAP client for connecting to and communicating with the webservice. <br>
 * encapsulates the SOAP client layer and can be used by e.g. an operator GUI. <br>
 * two main functions: <br>
 * forwarding commands to webservice and get results <br>
 * polling the webservice for events from device and inform listeners (e.g. operator GUI) <br>
 * it is the same client as the RzProtoOperatorClient2 but for Apache SOAP instead of Apache AXIS.
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoOperatorClient extends Observable implements Runnable {

        // the polling thread
        private Thread  thisOne;
        // shows status of the polling thread
        private boolean active;
        // polling intervall (in seconds)
        private int pollIntervall;                      // <---- config
        // soap server url
        private String soapServerURL;           // <---- config
        // web service id
        private String webServiceID;            // <---- config


        /**
         * The class constructor.
         *
         */
        public RzProtoOperatorClient() {
                System.out.println("<RzProtoOperatorClient> -> initializing ...");
                this.thisOne       = new Thread(this);
                this.active        = true;
                this.pollIntervall = 10;
                this.soapServerURL = "http://localhost:8080/soap/servlet/rpcrouter";
                this.webServiceID  = "urn:rzprotooperatorwebservice";
        }

        /**
         * thread starter method, periodically check for events by polling the webservice. <br>
         * if there are new events inform observer (e.g. GUI)
         *
         */
        public void run() {
                while (active) {
                        // check if there are new events, if neccessary inform listener (e.g. operator GUI)
                        try {
                                Vector events = sendEventRequest();
                                if (events.size() > 0) {
                                        setChanged();
                                        notifyObservers(events);
                                }
                        }
                        catch (Exception e) {
                                e.printStackTrace();
                        }

                        // wait <pollInterval> seconds for next call
                        try {
                                this.thisOne.sleep(this.pollIntervall*1000);
                        }
                        catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }

        /**
         * start the polling thread.
         */
        public void startup() {
                System.out.println("<RzProtoOperatorClient> -> start polling thread ...");
                this.thisOne.start();
        }

        /**
         * shutdown client by stopping the polling thread.
         *
         */
        public void shutdown() {
                active = false;
                System.out.println("<RzProtoOperatorClient> -> shutdown ...");
        }

        /**
         * send request for device status of the given device to webservice.
         *
         * @param   deviceName name of device in J/XFS layer (e.g. CashDispenser@localhost)
         * @return   device status as String (for GUI)
         * @exception        Exception
         */
        public String sendStatusRequest(String deviceName) throws Exception {
```

```
        // send rpc
        Vector params = new Vector();
        params.addElement(new Parameter("deviceName", String.class, deviceName,
        Constants.NS_URI_SOAP_ENC));
        DeviceStatus deviceStatus = (DeviceStatus)
        this.sendRPCAndUnmarshal(this.soapServerURL, this.webServiceID, "getDeviceStatus",
        true, params, DeviceStatus.class);
        // build response for GUI
        String res =  "Device State:\n" +
                      "is busy              = " + deviceStatus.getBusy() + "\n" +
                      "is claimed           = " + deviceStatus.getClaimed() + "\n" +
                      "is claimPending      = " + deviceStatus.getClaimPending() + "\n" +
                      "is hardware error    = " + deviceStatus.getHardwareError() + "\n" +
                      "is open              = " + deviceStatus.getOpen() + "\n" +
                      "is power save        = " + deviceStatus.getPowerSave() + "\n" +
                      "is user action error = " + deviceStatus.getUserActionError() + "\n" +
                      "is working           = " + deviceStatus.getWorking() + "\n\n";
        return res;
    }

    /**
     * send request for cassette counters of the given device to webservice.
     *
     * @exception       Exception
     * @param   deviceName name of device in J/XFS layer (e.g. CashDispenser@localhost)
     * @return    cassette counter (CashUnit) as String (for GUI)
     */
    public String sendCassetteCountersRequest(String deviceName) throws Exception {
        // send rpc
        Vector params = new Vector();
        params.addElement(new Parameter("deviceName", String.class, deviceName,
        Constants.NS_URI_SOAP_ENC));
        CashUnit cashUnit  = (CashUnit) this.sendRPCAndUnmarshal(this.soapServerURL,
        this.webServiceID, "getCassCount", true, params, CashUnit.class);
        // build response for GUI
        StringBuffer resBuf = new StringBuffer();
        resBuf.append("Cash Unit:\n");
        for (int i=0; i<cashUnit.getLogicalCashUnitCount(); i++) {
            LogicalCashUnit lcu = cashUnit.getLogicalCashUnit(i);
            resBuf.append("Logical Cash Unit <" + lcu.getId() + "> (");
            resBuf.append((lcu.getValue()/100) + " " + lcu.getCurrencyCode() + ")\n");
            resBuf.append("initial count = " + lcu.getInitialCount() + "\n");
            resBuf.append("count         = " + lcu.getCount() + "\n");
            resBuf.append("type          = " + lcu.getType() + "\n");
            resBuf.append("status        = " + lcu.getStatus() + "\n");
        }
        String res = resBuf.toString();
        return res;
    }

    /**
     * send request to webservice to register this client for events from J/XFS / JMX.
     *
     * @exception       Exception
     * @return    register successfull / not successfull
     */
    public String sendRegisterForNotificationsRequest() throws Exception {
        // send rpc
        Boolean response = (Boolean) this.sendRPCAndUnmarshal(this.soapServerURL,
        this.webServiceID, "registerForNotifications", false, null, Boolean.class);
        // if successfull start polling thread
        if (response.booleanValue()) {
            this.startup();
        }
        // response for GUI
        if (response.booleanValue()) {
            return "Registrierung erfolgreich!";
        }
        else {
            return "Registrierung nicht erfolgreich!";
        }
    }

    /**
     * send request to webservice to deregister this client for events from J/XFS / JMX.
     *
     * @exception       Exception
     * @return    deregister successfull / not successfull
     */
    public String sendDeregisterForNotificationsRequest() throws Exception {
        // send rpc
        Boolean response = (Boolean) this.sendRPCAndUnmarshal(this.soapServerURL,
        this.webServiceID, "deregisterForNotifications", false, null, Boolean.class);
        // if successfull stop polling thread
        if (response.booleanValue()) {
            active = false;
        }
        // response for GUI
        if (response.booleanValue()) {
            return "Deregistrierung erfolgreich!";
        }
        else {
            return "Deregistrierung nicht erfolgreich!";
        }
    }

    /**
     * send request to webservice to get all undelivered events.
     *
     * @exception       Exception
     * @return    vector of events
     */
    public Vector sendEventRequest() throws Exception {
        // send rpc
        Vector response = (Vector) this.sendRPCAndUnmarshal(this.soapServerURL,
        this.webServiceID, "getUndeliveredEvents", false, null, Vector.class);
        // response for GUI
        return response;
    }

    /**
     * send request to webservice to check if specified device exists.
     *
     * @param   deviceName name of device in J/XFS layer (e.g. CashDispenser@localhost)
     * @exception       Exception
     * @return    true if device exists
     */
    public boolean isDeviceExisting(String deviceName) throws Exception {
        // send rpc
        Vector params = new Vector();
        params.addElement(new Parameter("deviceName", String.class, deviceName,
        Constants.NS_URI_SOAP_ENC));
        Boolean response = (Boolean) this.sendRPCAndUnmarshal(this.soapServerURL,
        this.webServiceID, "isDeviceExisting", false, params, Boolean.class);
        return response.booleanValue();
    }


    /**
     * send RPC to web service.
     * TODO: generalize and outsource to soap helper class
     *
     * @param   soapServerUrl url of soap server (e.g. http://localhost:8080/soap/servlet/rpcrouter)
     * @param   webServiceID id of webservice (e.g. urn:mywebservice)
     * @param   methodName remote method to invoke
     * @param   xmlResponse specifies if the response is in xml literal style
     * @param   params vector of params for target method
     * @exception       SOAPException
     * @exception       MalformedURLException
     * @return    response object from soap server
     */
    private Object sendRPC( String soapServerUrl, String webServiceID,
                            String methodName, boolean xmlResponse, Vector params)
                            throws MalformedURLException, SOAPException {
        // build call
        Call call = new Call();
        call.setTargetObjectURI(webServiceID);
        call.setMethodName(methodName);
        if (xmlResponse) {
            call.setEncodingStyleURI(Constants.NS_URI_LITERAL_XML);
        }
        else {
            call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
        }
        call.setParams(params);
        URL soapServerURL = new URL(soapServerUrl);

        // send RPC, invoke method and get response
        System.out.println("<RzProtoOperatorClient> -> sending soap call to invoke <" +
        methodName + "> ...");
        Response response = call.invoke(soapServerURL, "");

        // extract RPC result out of response
        if (response.generatedFault()) {
            throw new SOAPException(response.getFault().getFaultCode(),
            response.getFault().getFaultString());
        }
        else {
            System.out.println("<RzProtoOperatorClient> -> soap call successfull ...");
            Parameter result    = response.getReturnValue();
            Object    rpcResult = result.getValue();
            return rpcResult;
        }
    }

    /**
     * send rpc and unmarhal to given type.
     * TODO: generalize and outsource to soap helper class
     *
     * @param   soapServerUrl url of soap server (e.g. http://localhost:8080/soap/servlet/rpcrouter)
     * @param   webServiceID id of webservice (e.g. urn:mywebservice)
     * @param   methodName remote method to invoke
     * @param   xmlResponse specifies if the response is in xml literal style
     * @param   params vector of params for target method
     * @param   responseType class to marshal to
     * @exception       MarshalException
     * @exception       SOAPException
     * @exception       ValidationException
     * @exception       MalformedURLException
     * @return          unmarhaled response object from soap server
     */
    private Object sendRPCAndUnmarshal(  String soapServerUrl, String webServiceID,
                                         String methodName, boolean xmlResponse,
                                         Vector params,
                                         Class responseType)
                                         throws MalformedURLException, SOAPException,
                                         ValidationException, MarshalException  {
        Object response     = null;
        // send rpc
        Object soapResponse = this.sendRPC(soapServerUrl, webServiceID, methodName,
        xmlResponse, params);
        // check response
        // soap fault
        if (soapResponse instanceof Fault) {
            throw new SOAPException("Soap Fault", ((Fault)soapResponse).getFaultString());
        }
        // success
        else {
            // response in xml format
            if (xmlResponse) {
                Element  xmlResult = (Element) soapResponse;
                response = this.unmarshalElement(xmlResult, responseType.getName());
                return response;
            }
            // response in simple format (e.g. String) -> no unmarhalling required
            else {
                response = soapResponse;
                return response;
            }
        }
    }

    /**
     * unmarshalling of given xmlElement to given unmarhalClass.
     * TODO: usage of java meta modell; generalize and  outsource to soap helper class
     *
     * @param   xmlElement xml element to be marshaled
     * @param   unmarshalClass class to marshal to
     * @exception       MarshalException
     * @exception       ValidationException
     * @return    unmarhaled element
     */
    private Object unmarshalElement(Element xmlElement, String unmarshalClass)
                                    throws ValidationException, MarshalException {
        System.out.println("<RzProtoOperatorClient> -> unmarshalling <XML element> to <" +
        unmarshalClass + "> ...");
        // element to string, because StringReader is desired
        DOM2Writer   dom2writer  = new DOM2Writer();
        String       xmlDocument = dom2writer.nodeToString(xmlElement);
        StringReader reader      = new StringReader(xmlDocument);
        // unmarshal to CashUnit
        if (unmarshalClass.equals("CashUnit") ||
        unmarshalClass.equals("com.wincornixdorf.pce.rzdiplomarbeit.schemata.
        cashunitschema.CashUnit")) {
            CashUnit cashUnit = CashUnit.unmarshal(reader);
            return cashUnit;
        }
        // unmarshal to DeviceStatus
```

```
            else if (unmarshalClass.equals("DeviceStatus") ||
            unmarshalClass.equals("com.wincornixdorf.pce.rzdiplomarbeit.schemata.
            devicestatusschema.DeviceStatus")) {
                  DeviceStatus deviceStatus = DeviceStatus.unmarshal(reader);
                  return deviceStatus;
            }
            // unknown
            else {
                  System.out.println("<RzProtoOperatorClient> -> <" + unmarshalClass + ">
                  unknown ...");
                  throw new MarshalException(unmarshalClass + " unknown!");
            }
      }
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.soap.operatorclient;

// castor framework classes for unmarshalling/marshalling from/to xml
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.cashunitschema.*;
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.devicestatusschema.*;

// J2SE 1.4
import java.util.ArrayList;
import java.net.URL;
import java.net.MalformedURLException;
import java.io.StringWriter;
import java.io.StringReader;
import java.util.Observable;
import java.rmi.RemoteException;
// Castor 0.9.3
import org.exolab.castor.xml.ValidationException;
import org.exolab.castor.xml.MarshalException;
// AXIS 1.0
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import javax.xml.rpc.JAXRPCException;
// Xerces 1.4.3
import org.w3c.dom.Element;
import org.w3c.dom.Document;
import org.apache.soap.util.xml.DOM2Writer;


/**
 * SOAP client for connecting to and communicating with the webservice. <br>
 * encapsulates the SOAP client layer and can be used by e.g. an operator GUI. <br>
 * two main functions: <br>
 * forwarding commands to webservice and get results <br>
 * polling the webservice for events from device and inform listeners (e.g. operator GUI) <br>
 * it is the same client as the RzProtoOperatorClient but for Apache AXIS instead of Apache SOAP.
 *
 * @author  Roger Zacharias
 * @version 1.0
 */
public class RzProtoOperatorClient2 extends Observable implements Runnable {

      // the polling thread
      private Thread  thisOne;
      // shows status of the polling thread
      private boolean active;
      // polling intervall (in seconds)
      private int pollIntervall;                  // <---- config
      // soap server url
      private String soapServerURL;               // <---- config
      // web service id
      private String webServiceID;                // <---- config


      /**
       * The class constructor.
       *
       */
      public RzProtoOperatorClient2() {
            System.out.println("<RzProtoOperatorClient2> -> initializing ...");
            this.thisOne       = new Thread(this);
            this.active        = true;
```

```
            this.pollIntervall = 10;
            this.soapServerURL = "http://localhost:8080/axis/servlet/AxisServlet";
            this.webServiceID  = "RzProtoOperatorWebService";
      }

      /**
       * thread starter method, periodically check for events by polling the webservice. <br>
       * if there are new events inform observer (e.g. GUI)
       *
       */
      public void run() {
            while (active) {
                  // check if there are new events, if neccessary inform listener (e.g. operator GUI)
                  try {
                        ArrayList events = sendEventRequest();
                        if (events.size() > 0) {
                              setChanged();
                              notifyObservers(events);
                        }
                  }
                  catch (Exception e) {
                        e.printStackTrace();
                  }

                  // wait <pollInterval> seconds for next call
                  try {
                        this.thisOne.sleep(this.pollIntervall*1000);
                  }
                  catch (InterruptedException e) {
                        e.printStackTrace();
                  }
            }
      }

      /**
       * start the polling thread.
       */
      public void startup() {
            System.out.println("<RzProtoOperatorClient2> -> start polling thread ...");
            this.thisOne.start();
      }

      /**
       * shutdown client by stopping the polling thread.
       *
       */
      public void shutdown() {
            active = false;
            System.out.println("<RzProtoOperatorClient2> -> shutdown ...");
      }

      /**
       * send request for device status of the given device to webservice.
       *
       * @param   deviceName name of device in J/XFS layer (e.g. CashDispenser@localhost)
       * @return    device status as String (for GUI)
       * @exception         Exception
       */
      public String sendStatusRequest(String deviceName) throws Exception {
            // send rpc
            DeviceStatus deviceStatus = (DeviceStatus)
            this.sendRPCAndUnmarshal(this.soapServerURL, this.webServiceID, "getDeviceStatus",
```

```
            true, "deviceName", deviceName, DeviceStatus.class);
            // build response for GUI
            String res =   "Device State:\n" +
                           "is busy                = " + deviceStatus.getBusy() + "\n" +
                           "is claimed             = " + deviceStatus.getClaimed() + "\n" +
                           "is claimPending        = " + deviceStatus.getClaimPending() + "\n" +
                           "is hardware error      = " + deviceStatus.getHardwareError() + "\n" +
                           "is open                = " + deviceStatus.getOpen() + "\n" +
                           "is power save          = " + deviceStatus.getPowerSave() + "\n" +
                           "is user action error = " + deviceStatus.getUserActionError() + "\n" +
                           "is working             = " + deviceStatus.getWorking() + "\n\n";
            return res;
      }

      /**
       * send request for cassette counters of the given device to webservice.
       *
       * @exception         Exception
       * @param   deviceName name of device in J/XFS layer (e.g. CashDispenser@localhost)
       * @return    cassette counter (CashUnit) as String (for GUI)
       */
      public String sendCassetteCountersRequest(String deviceName) throws Exception {
            // send rpc
            CashUnit cashUnit  = (CashUnit) this.sendRPCAndUnmarshal(this.soapServerURL,
            this.webServiceID, "getCassCount", true, "deviceName", deviceName, CashUnit.class);
            // build response for GUI
            StringBuffer resBuf = new StringBuffer();
            resBuf.append("Cash Unit:\n");
            for (int i=0; i<cashUnit.getLogicalCashUnitCount(); i++) {
                  LogicalCashUnit lcu = cashUnit.getLogicalCashUnit(i);
                  resBuf.append("Logical Cash Unit <" + lcu.getId() + "> (");
                  resBuf.append((lcu.getValue()/100) + " " + lcu.getCurrencyCode() + ")\n");
                  resBuf.append("initial count = " + lcu.getInitialCount() + "\n");
                  resBuf.append("count         = " + lcu.getCount() + "\n");
                  resBuf.append("type          = " + lcu.getType() + "\n");
                  resBuf.append("status        = " + lcu.getStatus() + "\n");
            }
            String res = resBuf.toString();
            return res;
      }

      /**
       * send request to webservice to register this client for events from J/XFS / JMX.
       *
       * @exception         Exception
       * @return    register successfull / not successfull
       */
      public String sendRegisterForNotificationsRequest() throws Exception {
            // send rpc
            Boolean response = (Boolean) this.sendRPCAndUnmarshal(this.soapServerURL,
            this.webServiceID, "registerForNotifications", false, null, null, Boolean.class);
            // if successfull start polling thread
            if (response.booleanValue()) {
                  this.startup();
            }
            // response for GUI
            if (response.booleanValue()) {
                  return "Registrierung erfolgreich!";
            }
            else {
                  return "Registrierung nicht erfolgreich!";
            }
```

```
	}

	/**
	 * send request to webservice to deregister this client for events from J/XFS / JMX.
	 *
	 * @exception	Exception
	 * @return	deregister successfull / not successfull
	 */
	public String sendDeregisterForNotificationsRequest() throws Exception {
		// send rpc
		Boolean response = (Boolean) this.sendRPCAndUnmarshal(this.soapServerURL,
		this.webServiceID, "deregisterForNotifications", false, null, null, Boolean.class);
		// if successfull stop polling thread
		if (response.booleanValue()) {
			active = false;
		}
		// response for GUI
		if (response.booleanValue()) {
			return "Deregistrierung erfolgreich!";
		}
		else {
			return "Deregistrierung nicht erfolgreich!";
		}
	}

	/**
	 * send request to webservice to get all undelivered events.
	 *
	 * @exception	Exception
	 * @return	vector of events
	 */
	public ArrayList sendEventRequest() throws Exception {
		// send rpc
		ArrayList response = (ArrayList) this.sendRPCAndUnmarshal(this.soapServerURL,
		this.webServiceID, "getUndeliveredEvents", false, null, null, ArrayList.class);
		// response for GUI
		return response;
	}

	/**
	 * send request to webservice to check if specified device exists.
	 *
	 * @param	deviceName name of device in J/XFS layer (e.g. CashDispenser@localhost)
	 * @exception	Exception
	 * @return	true if device exists
	 */
	public boolean isDeviceExisting(String deviceName) throws Exception {
		// send rpc
		Boolean response = (Boolean) this.sendRPCAndUnmarshal(this.soapServerURL,
		this.webServiceID, "isDeviceExisting", false, "deviceName", deviceName, Boolean.class);
		return response.booleanValue();
	}


	/**
	 * send RPC to web service.
	 * TODO: generalize and outsource to soap helper class
	 *
	 * @param	soapServerUrl url of soap server (e.g. http://localhost:8080/soap/servlet/rpcrouter)
	 * @param	webServiceID id of webservice (e.g. urn:mywebservice)
	 * @param	methodName remote method to invoke
	 * @param	xmlResponse specifies if the response is in xml literal style
	 * @param	params vector of params for target method
	 * @exception	SOAPException
	 * @exception	MalformedURLException
	 * @return	response object from soap server
	 */
	private Object sendRPC( String soapServerUrl, String webServiceID,
				String methodName, boolean xmlResponse,
				String param, String paramValue)
				 throws MalformedURLException, RemoteException,
				JAXRPCException {
		// build call
		Service service = new Service();
		Call call = (Call) service.createCall();
		call.setTargetEndpointAddress(new URL(soapServerUrl));
		call.setOperationName(methodName);
		call.setProperty(Call.NAMESPACE, webServiceID);
		Object[] paramValues = { paramValue };
		if (paramValue == null) {
			paramValues = null;
		}
		if (param != null) {
			call.addParameter(param, XMLType.XSD_STRING, Call.PARAM_MODE_IN);
		}

		// send RPC, invoke method and get response
		System.out.println("<RzProtoOperatorClient2> -> sending soap call to invoke <" +
		methodName + "> ...");
		Object	rpcResult = call.invoke(paramValues);
		System.out.println("<RzProtoOperatorClient2> -> soap call successfull ...");
		return rpcResult;
	}

	/**
	 * send rpc and unmarhal to given type.
	 * TODO: generalize and outsource to soap helper class
	 *
	 * @param	soapServerUrl url of soap server (e.g. http://localhost:8080/soap/servlet/rpcrouter)
	 * @param	webServiceID id of webservice (e.g. urn:mywebservice)
	 * @param	methodName remote method to invoke
	 * @param	xmlResponse specifies if the response is in xml literal style
	 * @param	params vector of params for target method
	 * @param	responseType class to marshal to
	 * @exception	MarshalException
	 * @exception	SOAPException
	 * @exception	ValidationException
	 * @exception	MalformedURLException
	 * @return	unmarhaled response object from soap server
	 */
	private Object sendRPCAndUnmarshal(	String soapServerUrl, String webServiceID,
						String methodName, boolean xmlResponse,
						String param, String paramValue,
						Class responseType)
						throws MalformedURLException, ValidationException,
						MarshalException, JAXRPCException,
						RemoteException	{
		Object response	= null;
		// send rpc
		Object soapResponse = this.sendRPC(soapServerUrl, webServiceID, methodName,
		xmlResponse, param, paramValue);

		// response in xml format
		if (xmlResponse) {
```

```
			Element	xmlResult = (Element) soapResponse;
			response = this.unmarshalElement(xmlResult, responseType.getName());
			return response;
		}
		// response in simple format (e.g. String) -> no unmarhalling required
		else {
			response = soapResponse;
			return response;
		}
	}

	/**
	 * unmarshalling of given xmlElement to given unmarhalClass.
	 * TODO: usage of java meta modell; generalize and	outsource to soap helper class
	 *
	 * @param	xmlElement xml element to be marshaled
	 * @param	unmarshalClass class to marshal to
	 * @exception	MarshalException
	 * @exception	ValidationException
	 * @return	unmarhaled element
	 */
	private Object unmarshalElement(Element xmlElement, String unmarshalClass)
					throws ValidationException, MarshalException {
		System.out.println("<RzProtoOperatorClient2> -> unmarshalling <XML element> to <" +
		unmarshalClass + "> ...");
		// element to string, because StringReader is desired
		DOM2Writer	dom2writer	= new DOM2Writer();
		String	xmlDocument = dom2writer.nodeToString(xmlElement);
		StringReader reader	= new StringReader(xmlDocument);
		// unmarshal to CashUnit
		if (unmarshalClass.equals("CashUnit") ||
		unmarshalClass.equals("com.wincornixdorf.pce.rzdiplomarbeit.schemata.
		cashunitschema.CashUnit")) {
			CashUnit cashUnit = CashUnit.unmarshal(reader);
			return cashUnit;
		}
		// unmarshal to DeviceStatus
		else if (unmarshalClass.equals("DeviceStatus") ||
		unmarshalClass.equals("com.wincornixdorf.pce.rzdiplomarbeit.schemata.
		devicestatusschema.DeviceStatus")) {
			DeviceStatus deviceStatus = DeviceStatus.unmarshal(reader);
			return deviceStatus;
		}
		// unknown
		else {
			System.out.println("<RzProtoOperatorClient2> -> <" + unmarshalClass + ">
			unknown ...");
			throw new MarshalException(unmarshalClass + " unknown!");
		}
	}
}
```

Web Service

```
package com.wincornixdorf.pce.rzdiplomarbeit.soap.operatorwebservice;

// castor framework classes for unmarshalling/marshalling from/to xml
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.devicestatusschema.*;
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.cashunitschema.*;

// J2SE 1.4
import java.net.Socket;
import java.net.SocketException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.StringWriter;
import java.io.ByteArrayInputStream;
import java.util.Vector;
import java.util.Observer;
import java.util.Observable;
// Xerces 1.4.3
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;


/**
 * WebService that exposes the management application interface. <br>
 * 6 WebService methods: <br>
 * - register for notifications <br>
 * - deregister for notifications <br>
 * - get undelivered events <br>
 * - check if device ist existing <br>
 * - get cassette counters <br>
 * - get device status <br>
 * socket client to jmx adaptor for sending requests to JXFS / JMX <br>
 * observer of RzProtoEventListener to get events from JXFS / JMX <br>
 * This web service runs fine in Apache SOAP but there are some problems in Apache Axis! <br>
 * TODO: Registry auslagern!
 *
 * @author	Roger Zacharias
 * @version 1.0
 */
public class RzProtoOperatorWebService implements Observer {

	// the event listener that listens for events from JXFS / JMX
	private RzProtoEventListener eventListener;
	// Event-Container (Registry)
	// TODO: Registry auslagern
	private Vector eventHistory;
	// Socket-Client
	private Socket clientSocket;
	// streams from and to Socket-Server
	private ObjectOutputStream toServer;
	private ObjectInputStream	fromServer;
	// socket port for event listener
	private int eventPort;		// <-- config
	// socket port
	private int commandPort;	// <-- config
	// socket host to jmx adaptor
```

```
    private String adaptorHost;           // <-- config


    /**
     * The class contructor.
     *
     */
    public RzProtoOperatorWebService() {
            System.out.println("<RzProtoOperatorWebService> -> initializing ...");
            this.eventHistory  = new Vector();
            this.eventPort     = 8099;
            this.eventListener = new RzProtoEventListener(eventPort);
            this.commandPort   = 8089;
            this.adaptorHost   = "localhost";
    }

    /**
     * register the client for events and start event listener to get events from JXFS / JMX.
     *
     * @return   result from jmx adaptor (true/false)
     */
    public Boolean registerForNotifications() {
            System.out.println("<RzProtoOperatorWebService> -> method <registerForNotifications>
            accessed ...");

            Boolean res = (Boolean) this.sendRequestToSocketServer("register");
            System.out.println("<RzProtoOperatorWebService> -> Result:");
            System.out.println(res);
            System.out.println("<RzProtoOperatorWebService> -> Sending result to <Client> ...");

            // successfull
            if (res.booleanValue()) {
                    // registering as observer to get events from client
                    eventListener.addObserver(this);
                    // start event listener to get events from adaptor
                    eventListener.startup();
            }
            return res;
    }

    /**
     * deregister the client for events and shutdown event listener.
     *
     * @return   result from jmx adaptor (true/false)
     */
    public Boolean deregisterForNotifications() {
            System.out.println("<RzProtoOperatorWebService> -> method
            <deregisterForNotifications> accessed ...");

            Boolean res = (Boolean) this.sendRequestToSocketServer("deregister");
            System.out.println("<RzProtoOperatorWebService> -> Result:");
            System.out.println(res);
            System.out.println("<RzProtoOperatorWebService> -> Sending result to <Client> ...");

            // if successfull stop event listener getting events from adaptor
            if (res.booleanValue()) {
               eventListener.shutdown();
            }
            return res;
    }

    /**
```

```
     * get all events that came from JXFS / JMX in since the last poll / the last request.
     *
     * @return   vector of undelivered events as strings
     */
    public Vector getUndeliveredEvents() {
            System.out.println("<RzProtoOperatorWebService> -> method <getUndeliveredEvent>
            accessed ...");
            Vector res = (Vector) this.eventHistory.clone();
            this.eventHistory.clear();
            System.out.println("<RzProtoOperatorWebService> -> returning events to <client> ...");
            System.out.println("<RzProtoOperatorWebService> -> deleting events ...");
            return res;
    }

    /**
     * checks if specified device exists in J/XFS environment.
     *
     * @param  deviceName name of device that should be checked
     * @return   true if device exists, otherwise false
     */
    public Boolean isDeviceExisting(String deviceName) {
            System.out.println("<RzProtoOperatorWebService> -> method <isDeviceExisting>
            accessed ...");

            Boolean res = (Boolean) this.sendRequestToSocketServer("isDeviceExisting:" +
            deviceName);
            System.out.println("<RzProtoOperatorWebService> -> Result:");
            System.out.println(res);
            System.out.println("<RzProtoOperatorWebService> -> Sending result to <Client> ...");
            return res;
    }

    /**
     * get cassette counters (CashUnit object) of specified device. <br>
     * return as xml literal, therefore marshal it first.
     *
     * @param  deviceName name of device
     * @return   cassette counters of specified device (as xml element)
     */
    public Element getCassCount(String deviceName) {
            System.out.println("<RzProtoOperatorWebService> -> method <getCassCount>
            accessed ...");

            CashUnit cu = (CashUnit) this.sendRequestToSocketServer("casscount:" + deviceName);

            Element res = this.marshalObject(cu);
            System.out.println("<RzProtoOperatorWebService> -> Sending <XML Stream> to
            <Client> ...");
            return res;
    }

    /**
     * get device status (DeviceStatus object) of specified device. <br>
     * return as xml literal, therefore marshal it first.
     *
     * @param  deviceName name of device
     * @return   device status of specified device (as xml element)
     */
    public Element getDeviceStatus(String deviceName) {
            System.out.println("<RzProtoOperatorWebService> -> method <getDeviceStatus>
            accessed ...");
```

```
            DeviceStatus ds = (DeviceStatus) this.sendRequestToSocketServer("status:" +
            deviceName);
            Element res = this.marshalObject(ds);
            System.out.println("<RzProtoOperatorWebService> -> Sending <XML Stream> to
            <Client> ...");
            return res;
    }

    /**
     * this method is automatically called by the event listener,
     * if an event came in. <br>
     * add event to event registry.
     *
     * @param  o Observable
     * @param  arg event object
     *
     */
    public void update(Observable o, Object arg) {
            System.out.println("<RzProtoOperatorWebService> -> Adding event to event history ...");
            this.eventHistory.add((String)arg);
    }

    /**
     * send request to jmx adaptor through socket.
     */
    private synchronized Object sendRequestToSocketServer(String request) {
            System.out.println("<RzProtoOperatorWebService> -> building socket connection (port" +
            commandPort + ") to <JMX adaptor> ...");
            try {
                    this.clientSocket = new Socket(adaptorHost, commandPort);
                    this.toServer     = new ObjectOutputStream(clientSocket.getOutputStream());
                    this.fromServer   = new ObjectInputStream(clientSocket.getInputStream());
            }
            catch (Exception e) {
                    System.out.println("<RzProtoOperatorWebService> -> connecting to jmx adaptor
                    failed!");
                    e.printStackTrace();
            }

            System.out.println("<RzProtoOperatorWebService> -> sending request <" + request + ">
            to <JMX adaptor> ...");
            // write to socket
            try {
                    toServer.writeObject(request);
                    toServer.flush();
                    toServer.reset();
            }
            catch (Exception e) {
                    System.out.println("<RzProtoOperatorWebService> -> writing to jmx adaptor
                    failed!");
                    e.printStackTrace();
            }

            System.out.println("<RzProtoOperatorWebService> -> reading result from <JMX adaptor>
            ...");
            // read from socket
            Object response = null;
            try {
                    response = fromServer.readObject();
            }
            catch (Exception e) {
                    System.out.println("<RzProtoOperatorWebService> -> reading from jmx adaptor
```

```
                    failed!");
                    e.printStackTrace();
            }

            // release resources
            try {
                    clientSocket.close();
                    toServer.close();
                    fromServer.close();
            }
            catch (Exception e) {
                    e.printStackTrace();
            }

            return response;
    }

    /**
     * marshal given object to xml element.
     */
    private Element marshalObject(Object o) {
            // marshalling DeviceState object
            System.out.println("<RzProtoOperatorWebService> -> Marshalling <" +
            o.getClass().getName() + "> to <XML Stream> ...");
            // marshal DeviceState object to a in memory string
            String xmlDocument = null;
            try {
                    StringWriter writer = new StringWriter();
                    if (o instanceof CashUnit) {
                            ((CashUnit) o).marshal(writer);
                    }
                    else if (o instanceof DeviceStatus) {
                            ((DeviceStatus) o).marshal(writer);
                    }
                    else {
                            System.out.println("<RzProtoOperatorWebService> -> Unknown object for
                            marshalling!");
                    }
                    xmlDocument = writer.getBuffer().toString();
            }
            catch (Exception e) {
                     System.out.println("<RzProtoOperatorWebService> -> Marshalling failed!");
                     e.printStackTrace();
            }

            // getting element out of string
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            ByteArrayInputStream is = new ByteArrayInputStream(xmlDocument.getBytes());
            Element root = null;
            Document doc = null;
            try {
                    DocumentBuilder builder = factory.newDocumentBuilder();
                    doc = builder.parse(is);
                    root = (Element) doc.getDocumentElement();
            } catch (Exception e) {
                    System.out.println("<RzProtoOperatorWebService> -> Marshalling failed!");
                    e.printStackTrace();
            }
            System.out.println("<RzProtoOperatorWebService> -> Result:\n" + xmlDocument);

            return root;
    }
```

```
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.soap.operatorwebservice;

// J2SE 1.4
import java.net.ServerSocket;
import java.net.Socket;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Observable;


/**
 * Listens for events from JXFS / JMX. <br>
 * socket server in listening thread, get events from jmx adaptor through socket. <br>
 * inform webservice of events.
 *
 * @author  Roger Zacharias
 * @version 1.0
 */
public class RzProtoEventListener extends Observable implements Runnable {

	// listening port to jmx adaptor
	private int port;
	// Socket-Server
	private ServerSocket serverSocket;
	// the listening thread
	private Thread  thisOne;
	// shows status of the listening thread
	private boolean active;

	/**
	 * The class constructor. <br>
	 *
	 * @param  port the port to listen for events
	 */
	public RzProtoEventListener(int port) {
		System.out.println("<RzProtoEventListener> -> initializing ...");
		this.port 		= port;
		this.serverSocket	= null;
		this.thisOne		= new Thread(this);
		this.active		= false;
	}

	/**
	 * thread starter method, starts server socket and wait for connections. <br>
	 * jmx adaptor delivers events through socket. <br>
	 * inform webservice if events came in.
	 *
	 */
	public void run() {
		this.active = true;
		try {
			System.out.println("<RzProtoEventListener> -> Binding socket listener to port (" +
			port + ") ...");
			serverSocket = new ServerSocket(port);
		}
		catch (Exception e) {
			e.printStackTrace();
		}

		System.out.println("<RzProtoEventListener> -> Listening for events ...");
		while (active) {
```

```
			try {
				Socket socket = serverSocket.accept();
				this.serviceClient(socket);
			}
			catch (Exception e) {
				e.printStackTrace();
			}
		}
	}

	/**
	 * start the listening thread.
	 *
	 */
	public void startup() {
		System.out.println("<RzProtoEventListener> -> start listening for events ...");
		this.thisOne.start();
	}

	/**
	 * shutdown the event listener by stopping the listening thread.
	 *
	 */
	public void shutdown() {
		System.out.println("<RzProtoEventListener> -> stop listening for events ...");
		this.active = false;
	}

	/**
	 * handle incoming event and notify observer (webservice)
	 */
	private void serviceClient (Socket clientConn) {
		System.out.println("<RzProtoEventListener> -> Incoming event ...");
		String 		event	= null;
		ObjectInputStream  fromClient = null;
		ObjectOutputStream toClient   = null;
		try {
			fromClient = new ObjectInputStream(clientConn.getInputStream());
			toClient   = new ObjectOutputStream(clientConn.getOutputStream());

			System.out.println("<RzProtoEventListener> -> Reading and identifying event from
			<RzProtoAdaptorServer> client ...");
			event = (String) fromClient.readObject();
			System.out.println("<RzProtoEventListener> -> Event is:\n" + event);

			// notify webservice of event and so add event to event history at webService
			setChanged();
			notifyObservers(event);

			// release resources
			clientConn.close();
			fromClient.close();
			toClient.close();
		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

JMX Agent

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.agent;

// MBeans to register: command forwarder, event forwarder and the SOAP adaptor
import com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans.RzProtoCommandForwarder;
import com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans.RzProtoEventForwarder;
import com.wincornixdorf.pce.rzdiplomarbeit.jmx.adaptor.RzProtoAdaptorServer;

// JDMK 4.2
import javax.management.ObjectName;
import javax.management.MBeanServer;
import javax.management.MBeanServerFactory;
import javax.management.MalformedObjectNameException;
import javax.management.NotCompliantMBeanException;
import javax.management.MBeanRegistrationException;
import javax.management.InstanceAlreadyExistsException;
import com.sun.jdmk.Trace;
import com.sun.jdmk.comm.HtmlAdaptorServer;


/**
 * JMX Agent - MAIN entry point. <br>
 * creates and manages the MBean server. <br>
 * registers the following MBean: <br>
 * - HTML Adaptor <br>
 * - SOAP Adaptor <br>
 * - Command Forwarder <br>
 * - Event Forwarder <br>
 *
 * @author  Roger Zacharias
 * @version 1.0
 */
public class RzProtoJmxAgent {

	// the MBean server
	private MBeanServer server;


	/**
	 * The class constructor. <br>
	 * creates the MBean Server.
	 *
	 */
	public RzProtoJmxAgent() {
		System.out.println("<RzProtoJmxAgent> -> Creating MBean server ...");
		this.server = MBeanServerFactory.createMBeanServer();
	}

	/**
	 * register given MBean with given object name at the MBean Server. <br>
	 *
	 * @param  mBean the MBean to register
	 * @param  objectName the object name of this MBean
	 */
	public void addMBean(Object mBean, String objectName) {
		System.out.println("<RzProtoJmxAgent> -> Registering MBean <" +
		mBean.getClass().getName() + "> ...");
		ObjectName mbDescription = null;
		try {
			mbDescription = new ObjectName(objectName);
```

```
            }
            catch (MalformedObjectNameException e) {
                    System.out.println("<RzProtoJmxAgent> -> Bad object name ...");
                    e.printStackTrace();
            }
            try {
                    server.registerMBean(mBean, mbDescription);
            }
            catch (NotCompliantMBeanException e) {
                    System.out.println("<RzProtoJmxAgent> -> MBean not ok ...");
                    e.printStackTrace();
            }
            catch (MBeanRegistrationException e) {
                    System.out.println("<RzProtoJmxAgent> -> MBean registration failed ...");
                    e.printStackTrace();
            }
            catch (InstanceAlreadyExistsException e) {
                    System.out.println("<RzProtoJmxAgent> -> MBean already exists ...");
                    e.printStackTrace();
            }
    }

    /**
     * MAIN for starting the JMX framework. <br>
     * create JMX Server the create MBeans and register them. <br>
     *
     * @param  args command line arguments
     */
    public static void main(String[] args) {
            RzProtoJmxAgent agent = new RzProtoJmxAgent();

            // register the Command Forwarder MBean
            System.out.println("\n<RzProtoJmxAgent> -> Creating <RzProtoCommandForwarder>
            ...");
            RzProtoCommandForwarder jxfsMBean = new RzProtoCommandForwarder();
            agent.addMBean(jxfsMBean,
            "DefaultDomain:name=RzProtoCommandForwarder,id=1,desc=command forwarder");

            // register Event Forwarder MBean
            System.out.println("\n<RzProtoJmxAgent> -> Creating <RzProtoEventForwarder> ...");
            RzProtoEventForwarder eventMBean = new RzProtoEventForwarder();
            agent.addMBean(eventMBean,
            "DefaultDomain:name=RzProtoEventForwarder,id=2,desc=event forwarder");

            // register and start the HTML adaptor MBean
            System.out.println("\n<RzProtoJmxAgent> -> Creating <HtmlAdaptorServer> ...");
            HtmlAdaptorServer htmlAdaptor = new HtmlAdaptorServer();
            agent.addMBean(htmlAdaptor, "Adaptor:name=html,port=8082");
            htmlAdaptor.start();

            // register and start the SOAP adaptor MBean
            System.out.println("\n<RzProtoJmxAgent> -> Creating <RzProtoAdaptorServer> ...");
            RzProtoAdaptorServer soapAdaptor = new RzProtoAdaptorServer(8089);
            agent.addMBean(soapAdaptor, "Adaptor:name=soap,port=8089");
            soapAdaptor.start();
    }
}
```

JMX Adapter

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.adaptor;

// J2SE 1.4
import java.net.Socket;


/**
 * defines the functionality of the RzProtoAdaptorServer. <br>
 *
 * @author        Roger Zacharias
 * @version       1.0
 */
public interface RzProtoAdaptorServerMBean {

        /**
         * deliver protocol of this protocol adaptor.
         *
         * @return        type of protocol of this adaptor (soap)
         */
        public String getProtocol();

        /**
         * return port on which this adaptor listens for client requests.
         *
         * @return        port on which this adaptor listens
         */
        public int getPort();

        /**
         * register a client (webservice) for notifications from J/XFS. <br>
         * this method is called by the RzProtoRequestHandler. <br>
         *
         * @param        client the client that wants to be notified
         */
        public void registerClient(Socket client);

        /**
         * deregister a client (webservice) for notifications from J/XFS. <br>
         * this method is called by the RzProtoRequestHandler. <br>
         *
         * @param        client the clients that wants to be deregistered
         *
         */
        public void deregisterClient(Socket client);
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.adaptor;

// JDMK 4.2
import com.sun.jdmk.comm.CommunicatorServer;
import com.sun.jdmk.comm.CommunicationException;
import javax.management.MBeanServer;
import javax.management.DynamicMBean;
import javax.management.NotificationListener;
import javax.management.Notification;
import javax.management.AttributeChangeNotification;
import javax.management.ObjectName;
// J2SE 1.4
import java.net.InetAddress;
import java.net.Socket;
import java.net.ServerSocket;
import java.net.SocketException;
import java.io.InterruptedIOException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Vector;
import java.util.Date;
import java.util.Enumeration;


/**
 * JMX adaptor Standard MBean for sending/receiving messages between SOAP framework and
 * JMX agent. <br>
 * called "SOAP adaptor". <br>
 * The super class defines (a lot of) generic behavior for the server part of a connector or an
 * adaptor. <br>
 * listens for client requests and creates threads for processing multiple request concurrently. <br>
 * listens to notifications from RzProtoEventMBean and forwards them to webservice -> client. <br>
 * this adaptor can be started/stopped with the inherited methods start()/stop(). <br>
 * when unregistered from MBean Server it is automatically stopped.
 *
 * @author  Roger Zacharias
 * @version 1.0
 */
public class RzProtoAdaptorServer extends CommunicatorServer implements NotificationListener {

        /*************** MBean attributes ***************/
        // port to listen on for client (webservice) requests
        private int listeningPort;                        // <-- config
        // Server-Socket to listen for client requests
        private ServerSocket  serverSocketListener;
        // socket representing a client
        private Socket clientSocket;
        // socket representing the registered client
        private Socket registeredClientSocket;
        // port of client (webservice) receiving events
        private int eventReceiverPort;                    // <-- config
        // host of client (webservice) receiving events
        private String eventReceiverHost;                 // <-- config



        /*************** MBean constructors ***************/
        /**
         * The class constructor.
         *
```

```
         * @param   port socket port to listen on for client requests
         */
        public RzProtoAdaptorServer(int port) {
                super(CommunicatorServer.HTTP_TYPE); // indicates that this is a HTTP_TYPE adaptor
                this.listeningPort          = port;
                this.serverSocketListener   = null;
                this.clientSocket           = null;
                this.registeredClientSocket = null;
                this.eventReceiverPort      = 8099;
                this.eventReceiverHost      = "localhost";
        }

        /*************** MBean methods ***************/
        /**
         * deliver protocol of this protocol adaptor.
         *
         * @return    type of protocol of this adaptor (soap)
         */
        public String getProtocol() {
                return "soap";
        }

        /**
         * return port on which this adaptor listens for client requests.
         *
         * @return    port on which this adaptor listens
         */
        public int getPort() {
                return this.listeningPort;
        }

        /**
         * binds socket listener to port to wait for requests from clients. <br>
         * then register this class for notifications from the JMX layer.
         *
         * @exception        CommunicationException
         * @exception        InterruptedException
         */
        protected void doBind() throws CommunicationException, InterruptedException {
                System.out.println("<RzProtoAdaptorServer> -> Binding socket listener to port ...");
                try {
                        serverSocketListener = new ServerSocket(listeningPort);
                }
                catch (SocketException e) {
                        if (e.getMessage().equals("Interrupted system call")) {
                                throw new InterruptedException(e.toString());
                        }
                        else {
                                throw new CommunicationException(e);
                        }
                }
                catch (InterruptedIOException e) {
                        throw new InterruptedException (e.toString());
                }
                catch (IOException e) {
                   throw new CommunicationException(e);
                }

                // register for notifications from JMX layer
                this.registerForJMXNotifications();
        }
```

```
        /**
         * listens for client requests. <br>
         * blocks until a client is available.
         *
         * @exception       CommunicationException
         * @exception       InterruptedException
         */
        protected void doReceive() throws CommunicationException, InterruptedException {
                System.out.println("<RzProtoAdaptorServer> -> Listening for requests ...");
                try {
                        clientSocket = serverSocketListener.accept();
                        System.out.println("<RzProtoAdaptorServer> -> Incoming request ...");
                }
                catch (SocketException e) {
                        if (e.getMessage().equals("<RzProtoAdaptorServer> -> Interrupted system call")) {
                                throw new InterruptedException(e.toString());
                        }
                        else {
                                throw new CommunicationException(e);
                        }
                }
                catch (InterruptedIOException e) {
                        throw new InterruptedException (e.toString());
                }
                catch (IOException e) {
                   throw new CommunicationException(e);
                }
        }

        /**
         * processes the request by creating a new thread and delegating the request to it. <br>
         * automatically called after doReceive().
         *
         * @exception       CommunicationException
         * @exception       InterruptedException
         */
        protected void doProcess() throws CommunicationException, InterruptedException {
                System.out.println("<RzProtoAdaptorServer> -> Sending request to
                <RzProtoRequestHandler> ...");
                // forward request to client handler
                RzProtoRequestHandler requestHandler = new RzProtoRequestHandler(clientSocket,
                this, super.getMBeanServer());
                // release resources
                requestHandler = null;
                clientSocket   = null;
        }

        /**
         * processes an error.
         *
         * @param   e an Exception
         * @exception       CommunicationException
         */
        protected void doError(Exception e) throws CommunicationException {
                throw new CommunicationException(e);
        }

        /**
         * close socket when connector goes offline.
         *
         * @exception       CommunicationException
         * @exception       InterruptedException
```

```
         */
        protected void doUnbind() throws CommunicationException, InterruptedException {
                System.out.println("<RzProtoAdaptorServer> -> Unbinding socket listener ...");
                try {
                        serverSocketListener.close();
                }
                catch (SocketException e) {
                        if (e.getMessage().equals("Interrupted system call")) {
                                throw new InterruptedException(e.toString());
                        }
                        else {
                                throw new CommunicationException(e);
                        }
                }
                catch (InterruptedIOException e) {
                        throw new InterruptedException (e.toString());
                }
                catch (IOException e) {
                   throw new CommunicationException(e);
                }
        }

        /**
         * implementation of the notification listener interface. <br>
         * method is called if an notification from the JMX layer came in. <br>
         * if a notification from the RzProtoEventMBean came in, <br>
         * forward it to the webservice -> client.
         *
         * @param   notification the notification from the JMX layer
         * @param   handback a handback object (no need here)
         */
        public void handleNotification(Notification notification, Object handback) {
                System.out.println("<RzProtoAdaptorServer> -> Received notification from JMX ...");
                String type = notification.getType();
                try {
                        // notification is from RzProtoEventMBean
                        if (type.equals(AttributeChangeNotification.ATTRIBUTE_CHANGE)) {
                                AttributeChangeNotification acNotification = (AttributeChangeNotification)
                                notification;
                                System.out.println("Date           = " + new
                                Date(acNotification.getTimeStamp()));
                                System.out.println("Attribute name = " + acNotification.getAttributeName());
                                System.out.println("Message        = " + acNotification.getMessage());
                                System.out.println("Old value      = " + acNotification.getOldValue());
                                System.out.println("New value      = " + acNotification.getNewValue());

                                String event = (new Date(acNotification.getTimeStamp())).toString() + " -> "
                                                + acNotification.getNewValue();

                                // if a client (webservice) is registered, send this event to it
                                if (this.registeredClientSocket != null) {
                                        System.out.println("<RzProtoAdaptorServer> -> Sending notification
                                        to <WebService> ...");
                                        Socket socket = new Socket(eventReceiverHost, eventReceiverPort);
                                        ObjectOutputStream toServer   = new
                                        ObjectOutputStream(socket.getOutputStream());
                                        ObjectInputStream  fromServer = new
                                        ObjectInputStream(socket.getInputStream());
                                        toServer.writeObject(event);
                                }
                        }
                        else {
```

```
                                System.out.println("<RzProtoAdaptorServer> -> Unknown event ...");
                        }
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
        }

        /**
         * register a client (webservice) for notifications from J/XFS. <br>
         * this method is called by the RzProtoRequestHandler. <br>
         * TODO: list of clients (webservices)
         *
         * @param   client client socket to register
         */
        public void registerClient(Socket client) {
                System.out.println("<RzProtoAdaptorServer> -> Registering client <" +
                client.getInetAddress() + "> for notifications ...");
                this.registeredClientSocket = client;
        }

        /**
         * deregister a client (webservice) for notifications from J/XFS. <br>
         * this method is called by the RzProtoRequestHandler. <br>
         * TODO: list of clients (webservices)
         *
         * @param   client client socket to deregister
         *
         */
        public void deregisterClient(Socket client) {
                System.out.println("<RzProtoAdaptorServer> -> Deregistering client <" +
                client.getInetAddress() + "> for notifications ...");
                this.registeredClientSocket = null;
        }

        /**
         * register this adaptor for notifications from the RzProtoEventMBean. <br>
         * to receive events from J/XFS.
         */
        private void registerForJMXNotifications() {
                System.out.println("<RzProtoAdaptorServer> -> Registering for notifications from
                <RzProtoEventMBean> ...");
                try {
                        ObjectName objectName = new
                        ObjectName("DefaultDomain:name=RzProtoEventForwarder,id=2,desc=event
                        forwarder");
                        this.getMBeanServer().addNotificationListener(objectName, this, null, null);
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.adaptor;

// cassette counters schema for castor framework
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.cashunitschema.*;
// device status schema for castor framework
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.devicestatusschema.*;

// JDMK 4.2
import com.sun.jdmk.comm.CommunicatorServer;
import javax.management.MBeanServer;
import javax.management.ObjectName;
// J2SE 1.4
import java.net.Socket;
import java.net.SocketException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;


/**
 * handles incoming requests from the soap adaptor. <br>
 * connects to client (webservice) and send/receive messages. <br>
 * supports 5 types of requests: <br>
 * - request for register <br>
 * - request for deregister <br>
 * - request for cassette counters <br>
 * - request for device status <br>
 * - request for is device existing <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoRequestHandler extends Thread {

        // the service requesting client
        private Socket sockClient;
        // the adaptor server connected to this request handler
        private RzProtoAdaptorServer adaptorServer;
        // the MBean server to which the adaptor is connected
        private MBeanServer mbServer;
        // streams from and to client (webservice)
        private ObjectOutputStream toClient;
        private ObjectInputStream  fromClient;


        /**
         * The class constructor. <br>
         * starting the thread to process the request and generate the response. <br>
         *
         * @param   s the client socket
         * @param   adaptorServer the connected adaptor server
         * @param   mbServer the MBean Server to which the adaptor is connected
         */
        public RzProtoRequestHandler(Socket s, RzProtoAdaptorServer adaptorServer, MBeanServer
                                        mbServer) {
                this.sockClient    = s;
                this.adaptorServer = adaptorServer;
                this.mbServer      = mbServer;
                this.toClient      = null;
                this.fromClient    = null;
                this.run();
```

```
	}

	/**
	 * thread starter for processing client request. <br>
	 */
	public void run() {
		this.processRequest();
	}

	/**
	 * build connection from/to client. <br>
	 * read and identify request. <br>
	 * process the request and generate response by getting information from MBeans. <br>
	 */
	private void processRequest() {
		System.out.println("<RzProtoRequestHandler> -> Processing
		<RzProtoOperatorWebService> client request ...");
		String		clientRequest = null;
		try {
			fromClient = new ObjectInputStream(sockClient.getInputStream());
			toClient   = new ObjectOutputStream(sockClient.getOutputStream());

			System.out.println("<RzProtoRequestHandler> -> Reading and identifying request
			from <RzProtoOperatorWebService> client ...");
			clientRequest = (String) fromClient.readObject();
			System.out.println("<RzProtoRequestHandler> -> Request is:\n" + clientRequest);
		}
		catch (Exception e) {
			e.printStackTrace();
		}

		// identify type of request
		// request for register
		if (clientRequest.equals("register")) {
			// register client for notifications
			System.out.println("<RzProtoRequestHandler> -> Request for <notification-
			registering> received ...");
			// return "Registration successful!" to web service
			System.out.println("<RzProtoRequestHandler> -> Return \"Registration
			successful!\" to <RzProtoOperatorWebService> ...");
			// runtime workaround
			try { Thread.sleep(500); } catch (Exception e) {}
			this.sendClientResult(new Boolean(true));
			// register this client for notifications
			this.adaptorServer.registerClient(this.sockClient);
		}
		// request for deregister
		else if (clientRequest.equals("deregister")) {
			// deregister client for notifications
			System.out.println("<RzProtoRequestHandler> -> Request for <deregistering>
			received ...");
			// return "Deregistration successful!" to web service
			System.out.println("<RzProtoRequestHandler> -> Return \"Deregistration
			successful!\" to <RzProtoOperatorWebService> ...");
			// runtime workaround
			try { Thread.sleep(500); } catch (Exception e) {}
			this.sendClientResult(new Boolean(true));
			// deregister this client for notifications
			this.adaptorServer.deregisterClient(this.sockClient);
		}
		// request for cassette counters
		else if (clientRequest.startsWith("casscount")) {
			System.out.println("<RzProtoRequestHandler> -> Request for <CashUnit>
			received ...");
			String deviceName = clientRequest.substring(clientRequest.indexOf(":")+1);
			// invoke method on MBean to get cash unit object
			System.out.println("<RzProtoRequestHandler> -> Sending request to
			<RzProtoCommandForwarder> ...");
			CashUnit result = (CashUnit) this.invokeMBeanMethod("getCashUnitObject",
			deviceName);
			System.out.println("<RzProtoRequestHandler> -> Response from
			<RzProtoCommandForwarder>: \n" + result);
			// return MBean result to web service
			System.out.println("<RzProtoRequestHandler> -> Return response from
			<RzProtoCommandForwarder> to <RzProtoOperatorWebService> ..." + result);
			// runtime workaround
			try { Thread.sleep(500); } catch (Exception e) {}
			this.sendClientResult(result);
		}
		// request for device status
		else if (clientRequest.startsWith("status")) {
			System.out.println("<RzProtoRequestHandler> -> Request for <DeviceStatus>
			received ...");
			String deviceName = clientRequest.substring(clientRequest.indexOf(":")+1);
			// invoke method on MBean to get device status object
			System.out.println("<RzProtoRequestHandler> -> Sending request to
			<RzProtoCommandForwarder> ...");
			DeviceStatus result = (DeviceStatus)
			this.invokeMBeanMethod("getDeviceStatusObject", deviceName);
			System.out.println("<RzProtoRequestHandler> -> Response from
			<RzProtoCommandForwarder>: \n" + result);
			// return MBean result to web service
			System.out.println("<RzProtoRequestHandler> -> Return response from
			<RzProtoCommandForwarder> to <RzProtoOperatorWebService> ..." + result);
			// runtime workaround
			try { Thread.sleep(500); } catch (Exception e) {}
			this.sendClientResult(result);
		}
		// request for is device existing
		else if (clientRequest.startsWith("isDeviceExisting")) {
			System.out.println("<RzProtoRequestHandler> -> Request for <isDeviceExisting>
			received ...");
			String deviceName = clientRequest.substring(clientRequest.indexOf(":")+1);
			// invoke method on MBean to get info about device existence
			System.out.println("<RzProtoRequestHandler> -> Sending request to
			<RzProtoCommandForwarder> ...");
			Boolean result = (Boolean) this.invokeMBeanMethod("isDeviceExisting",
			deviceName);
			System.out.println("<RzProtoRequestHandler> -> Response from
			<RzProtoCommandForwarder>: \n" + result);
			// return MBean result to web service
			System.out.println("<RzProtoRequestHandler> -> Return response from
			<RzProtoCommandForwarder> to <RzProtoOperatorWebService> ..." + result);
			// runtime workaround
			try { Thread.sleep(500); } catch (Exception e) {}
			this.sendClientResult(result);
		}
	}


	/**
	 * write result object to client.
	 */
```

```
	private void sendClientResult(Object result) {
		try {
			toClient.writeObject(result);
			toClient.flush();
			toClient.reset();
			sockClient.close();
			fromClient.close();
			toClient.close();
		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}

	/**
	 * invoke method on RzProtoCommandForwarder MBean
	 */
	private Object invokeMBeanMethod(String operationName, String deviceName) {
		Object result = null;
		try {
			ObjectName mbeanName     = new
			ObjectName("DefaultDomain:name=RzProtoCommandForwarder,
			id=1,desc=command forwarder");
			Object[]   params        = {deviceName};
			String[]   signature     = {"java.lang.String"};
			result = mbServer.invoke(mbeanName, operationName, params, signature);
		}
		catch (Exception e) {
			e.printStackTrace();
		}
		return result;
	}
}
```

JMX MBeans

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans;

// cassette counters and device status schema for castor framework
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.cashunitschema.*;
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.devicestatusschema.*;


/**
 * defines the functionality of the RzProtoCommandForwarder. <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public interface RzProtoCommandForwarderMBean {

	/**
	 * get the number of requests that occurred.
	 *
	 * @return    number of occurred requests
	 */
	public int getRequestNumber();

	/**
	 * reset the number of requests that occurred.
	 *
	 */
	public void resetRequestNumber();

	/**
	 * return an (already mapped) CashUnit object received from J/XFS layer.
	 *
	 * @param   deviceName name of device
	 * @return    already mapped CashUnit object
	 */
	public CashUnit getCashUnitObject(String deviceName);

	/**
	 * return an (already mapped) DeviceStatus object received from J/XFS layer.
	 *
	 * @param   deviceName name of device
	 * @return    already mapped DeviceStatus object
	 */
	public DeviceStatus getDeviceStatusObject(String deviceName);

	/**
	 * return true if given device exists, after querying the J/XFS layer.
	 *
	 * @param   deviceName name of device
	 * @return    true if device exists, otherwise false
	 */
	public Boolean isDeviceExisting(String deviceName);
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans;

// cassette counters and device status schema for castor framework
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.cashunitschema.*;
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.devicestatusschema.*;
// J/XFS 2.0
import
com.wincornixdorf.pce.rzdiplomarbeit.jxfs.cassettecountersreader.RzProtoCassetteCountersReader;
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.statusreader.RzProtoJxfsStatusReader;


/**
 * Standard MBean as a proxy to the J/XFS layer. <br>
 * three main functions: <br>
 * - returns cassette counters of a given device <br>
 * - returns device status of a given device <br>
 * - checks if device with given name exists <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoCommandForwarder implements RzProtoCommandForwarderMBean {

	/*************** MBean attributes ***************/
	// number of requests that came in
	private int requestNumber;
	// the cassette counters reader
	private RzProtoCassetteCountersReader cassCountReader;
	// the status reader
	private RzProtoJxfsStatusReader statusReader;


	/*************** MBean constructors ***************/
	/**
	 * The class constructor. <br>
	 * starts the readers.
	 *
	 */
	public RzProtoCommandForwarder() {
		this.requestNumber   = 0;
		this.cassCountReader = new RzProtoCassetteCountersReader();
		this.statusReader    = new RzProtoJxfsStatusReader();
		// start readers
		cassCountReader.startup();
		statusReader.startup();
	}

	/*************** MBean methods ***************/
	/**
	 * returns the number of occurred requests.
	 *
	 * @return    number of occurred requests
	 */
	public int getRequestNumber() {
		return this.requestNumber;
	}

	/**
	 * resets the number of occurred requests.
	 *
	 */
```

```
	public void resetRequestNumber() {
		this.requestNumber = 0;
	}

	/**
	 * return an (already mapped) CashUnit object received from J/XFS layer. <br>
	 * use the RzProtoCassetteCountersReader to get this information.
	 *
	 * @param   deviceName name of device
	 * @return    already mapped CashUnit object
	 *
	 */
	public CashUnit getCashUnitObject(String deviceName) {
		this.requestNumber++;

		System.out.println("<RzProtoCommandForwarder> -> Getting request for <CashUnit>
		...");
		// get J/XFS info
		System.out.println("<RzProtoCommandForwarder> -> Sending request to
		<RzProtoCassetteCountersReader> ...");
		cassCountReader.startup();
		System.out.println("<RzProtoCommandForwarder> -> Receiving result from
		<RzProtoCassetteCountersReader> ...");
		CashUnit cu = cassCountReader.getCassettes(deviceName);
		//cassCountReader.shutdown();
		System.out.println("<RzProtoCommandForwarder> -> Sending result to <JMX MBean
		Server> ...");
		return cu;
	}

	/**
	 * return an (already mapped) DeviceStatus object received from J/XFS layer. <br>
	 * use the RzProtoJxfsStatusReader to get this information.
	 *
	 * @param   deviceName name of device
	 * @return    already mapped DeviceStatus object
	 */
	public DeviceStatus getDeviceStatusObject(String deviceName) {
		this.requestNumber++;

		System.out.println("<RzProtoCommandForwarder> -> Getting request for
		<DeviceStatus> ...");
		// get J/XFS info
		System.out.println("<RzProtoCommandForwarder> -> Sending request to
		<RzProtoJxfsStatusReader> ...");
		System.out.println("<RzProtoCommandForwarder> -> Receiving result from
		<RzProtoJxfsStatusReader> ...");
		DeviceStatus ds = statusReader.getStatus(deviceName);
		System.out.println("<RzProtoCommandForwarder> -> Sending result to <JMX MBean
		Server> ...");
		return ds;
	}

	/**
	 * return true if given device exists, after querying the J/XFS layer. <br>
	 * use the RzProtoJxfsStatusReader to get this information.
	 *
	 * @param   deviceName name of device
	 * @return    true if device exists, otherwise false
	 */
	public Boolean isDeviceExisting(String deviceName) {
		this.requestNumber++;
```

```
		System.out.println("<RzProtoCommandForwarder> -> Getting request for
		<isDeviceExisting> ...");
		// get J/XFS info
		System.out.println("<RzProtoCommandForwarder> -> Sending request to
		<RzProtoJxfsStatusReader> ...");
		System.out.println("<RzProtoCommandForwarder> -> Receiving result from
		<RzProtoJxfsStatusReader> ...");
		boolean exists = statusReader.isDeviceExisting(deviceName);
		System.out.println("<RzProtoCommandForwarder> -> Sending result to <JMX MBean
		Server> ...");
		return new Boolean(exists);
	}
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans;

/**
 * defines the functionality of the RzProtoEventForwarder. <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public interface RzProtoEventForwarderMBean {

	/**
	 * returns the number of occurred events.
	 *
	 * @return    number of occurred events
	 *
	 */
	public int getEventCount();

}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans;

// J/XFS connector
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.connector.RzProtoJxfsConnector;
// J/XFS event generator
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.eventgenerator.RzProtoJxfsEventGenerator;
// J/XFS 2.0
import com.jxfs.events.IJxfsStatusListener;
import com.jxfs.events.JxfsStatusEvent;
import com.jxfs.general.JxfsThresholdStatus;
import com.jxfs.control.cdr.JxfsLogicalCashUnit;
// JDMK 4.2
import javax.management.NotificationBroadcasterSupport;
import javax.management.AttributeChangeNotification;
// J2SE 1.4
import java.util.Date;


/**
 * Standard MBean as a proxy to the J/XFS layer. <br>
 * main functions: <br>
 * - inform listeners (adaptor -> webservice -> client) of events <br>
 * The super class NotificationBroadcasterSupport is an implementation of NotificationBroadcaster
 * interface. <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoEventForwarder extends NotificationBroadcasterSupport
                                    implements RzProtoEventForwarderMBean,
                                      IJxfsStatusListener {

        /*************** MBean attributes ***************/
        // name of observed device
        private String deviceName;
        // number of occurred events
        private int eventCount;
        // event that came in
        private String event;


        /*************** MBean constructors ***************/
        /**
         * The class constructor. <br>
         * start event generator.
         *
         */
        public RzProtoEventForwarder() {
                this.eventCount  = 0;
                this.deviceName  = "Dummy_CashDispenser@localhost";  // <-- config
                this.event       = "";
                RzProtoJxfsEventGenerator eventGenerator
                = new RzProtoJxfsEventGenerator(this, deviceName);
                eventGenerator.startup();
        }

        /*************** MBean methods ***************/
        /**
         * returns the number of occurred events.
         *
         * @return    number of occurred events
         */
        public int getEventCount() {
                return this.eventCount;
        }

        /**
         * listener interface method to handle StatusEvents. <br>
         * called by the RzProtoJxfsEventGenerator if a CashUnit is empty. <br>
         * then inform listeners.
         *
         * @param   event an status event
         *
         */
        public synchronized void statusOccurred(JxfsStatusEvent event) {
                if (event.getStatus() == JxfsThresholdStatus.JXFS_S_BIN_EMPTY) {
                        JxfsLogicalCashUnit lcu = (JxfsLogicalCashUnit) event.getDetails();
                        this.informListeners("JxfsATM - cassette <" + lcu.getUnitID() + "> empty");
                }
        }

        /**
         * if event occurred inform all listeners.
         */
        private void informListeners(String newEvent) {
                Date actualDate = new Date();
                long timeStamp  = actualDate.getTime();
                // send notification to all listeners
                // params: source, sequence number, time stamp, message,
                // attribute name, attribute type, old value, new value
                AttributeChangeNotification notification
                = new AttributeChangeNotification(this, eventCount, timeStamp, "device state changed",
                                                  "deviceState", "String", event, newEvent);
                sendNotification(notification);
                this.event = newEvent;
                this.eventCount++;
        }
}
```

J/XFS Connector

```
package com.wincornixdorf.pce.rzdiplomarbeit.jxfs.connector;

// J/XFS 2.0
import com.jxfs.general.*;       // contains classes used in more than one other package
import com.jxfs.control.*;       // contains all the interfaces to J/XFS Device Controls
import com.jxfs.control.alm.*;   // Device Control for a Alaram Device
import com.jxfs.control.msd.*;   // Device Control for a Magnetic Stripe Reader and Writer device
import com.jxfs.events.*;        // contains all the defined J/XFS events, exceptions and respective
listeners
import com.jxfs.forum.logging.*; // Logging support for J/XFS forum implementation of J/XFS kernel
// J2SE 1.4
import java.io.*;
import java.util.*;


/**
 * Class to connect to J/XFS. <br>
 * startup and shutdown the device manager and logger. <br>
 * singleton of this class holds manager and logger.
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoJxfsConnector {

        // the singleton connector
        static private RzProtoJxfsConnector singleton = null;
        // the device manager - the central controlling instance
        private JxfsDeviceManager manager;
        // the logger - for tracing and error logging
        private JxfsLogger logger;
        // counter for number of startups
        private int startupCount = 0;
        // name of property file
        private String initPropertyFileName  = "c:\\jxfsconnector.properties";
        // values to init the device manager
        private String Jxfs_sdm_name;
        private String Jxfs_workstation_name;
        private String Jxfs_config_key;
        private String Jxfs_server_port;
        private String Jxfs_server_name;
        // values to init the logger
        private String Jxfs_logFilename;
        private int    Jxfs_traceLevel;


        /**
         * private constructor for singleton object.
         * initializing the device manager and logger.
         */
        private RzProtoJxfsConnector() {
                System.out.println("<RzProtoJxfsConnector> -> initializing ...");

                manager  = JxfsDeviceManager.getReference();
                logger   = JxfsLogger.getReference();

                // get properties
                this.getProps();
                System.out.println("<RzProtoJxfsConnector> -> initialized with:");
                System.out.println("WORKSTATION NAME : " + Jxfs_workstation_name);
                System.out.println("SERVER      NAME : " + Jxfs_server_name);
                System.out.println("SERVER      PORT : " + Jxfs_server_port + "\n");
        }

        /**
         * get singleton connector object. <br>
         * create one if there isn´t one.
         *
         * @return    singleton object of PvJxfsConnector
         */
        static public RzProtoJxfsConnector getDefaultInstance() {
                if (singleton == null) {
                        singleton = new RzProtoJxfsConnector();
                        return singleton;
                }
                else return singleton;
        }

        /**
         * start the application and the device manager (only the first time). <br>
         * register application at the logger.
         *
         */
        public void startup() {
                System.out.println("<RzProtoJxfsConnector> -> starting ...");

                synchronized (this) {
                        startupCount++;
                        if (startupCount > 1) return;
                }
                try {
                        String configurationParameters = Jxfs_sdm_name + "," +
                        Jxfs_workstation_name + "," +
                        Jxfs_config_key + "," +
                        Jxfs_server_port + ";" +
                        Jxfs_server_name;
                        manager.initialize(configurationParameters);

                        StdLogFilter filter      = new StdLogFilter();
                        StdLoggerFile stdLogFile = new StdLoggerFile();
                        if (stdLogFile.initialize(Jxfs_logFilename) != true) {
                                System.out.println("<RzProtoJxfsConnector> -> Error during StdLoggerFile
                                initialization for" + Jxfs_logFilename + "!");
                        }
                        logger.addFilter(filter);
                        logger.addLogListener(stdLogFile);
                        logger.registerModule(Jxfs_config_key, "J/XFS-Connector");
                }
                catch(JxfsException exception) {
                        System.out.println("<RzProtoJxfsConnector> -> Error during DeviceManager and
                        Logger initialization!\n" + exception);
                        writeToLog("<RzProtoJxfsConnector> -> Error during DeviceManager and Logger
                        initialization!" + exception.toString());
                }

                writeToLog("<RzProtoJxfsConnector> -> Initialization of JxfsDeviceManager and
                JxfsLogger successfull!");
                System.out.println("<RzProtoJxfsConnector> -> JxfsDeviceManager and JxfsLogger
                started ...");
        }
```

```
    /**
     * stop the application and the device manager (only the last client). <br>
     * release and close the given device controls. <br>
     * deregister logger.
     *
     * @param   controls vector of device controls that should be closed
     */
    public void shutdown(Vector controls) {
        System.out.println("<RzProtoJxfsConnector> -> stopping ...");

        JxfsBaseControl control;
        JxfsStatus      status;
        if (controls != null) {
            for (Enumeration e = controls.elements(); e.hasMoreElements() ;) {
                control = (JxfsBaseControl) (e.nextElement());
                try {
                    status = control.getStatus();
                    if (status.isClaimed()) {
                        control.release(2000);
                    }
                    if (status.isOpen()) {
                        control.close();
                    }
                }
                catch (JxfsException exception) {
                    System.out.println("<RzProtoJxfsConnector> -> Release/Close of
                    device control failed!\n" + exception.toString());
                }
            }
        }
        synchronized (this) {
            startupCount--;
            if (startupCount > 0) return;
        }
        writeToLog("<RzProtoJxfsConnector> -> JxfsDeviceManager shutdown!");
        manager.shutdown();
        try {
            Thread.currentThread().sleep(5000);
        }
        catch (InterruptedException ign) {
            System.out.println(ign.toString());
        }
        logger.deregisterModule(Jxfs_config_key);

        System.out.println("<RzProtoJxfsConnector> -> JxfsDeviceManager shutdown ...");
        System.out.println("<RzProtoJxfsConnector> -> stopped ...");
    }

    /**
     * get singleton device manager.
     *
     * @return    the J/XFS device manager
     */
    public JxfsDeviceManager getJxfsDeviceManager() {
        return manager;
    }

    /**
     * get singleton logger.
     *
     * @return    the J/XFS logger
```

```
     */
    public JxfsLogger getJxfsLogger() {
        return logger;
    }

    /**
     * writes a message to the J/XFS log-file using the J/XFS logger.
     *
     * @param   msg message which should be written to the log-file
     *
     */
    public void writeToLog(String msg) {
        logger.writeLog (this, Jxfs_config_key, Jxfs_traceLevel, msg);
    }

    /**
     * read the property file and get the properties
     */
    private void getProps() {
        Properties initProps = new Properties();
        try {
            FileInputStream in = new FileInputStream(initPropertyFileName);
            initProps.load(in);
            in.close();
        }
        catch (Exception e) {
            System.out.println(e);
        }

        Jxfs_sdm_name          = initProps.getProperty("Jxfs_sdm_name");
        Jxfs_workstation_name  = initProps.getProperty("Jxfs_workstation_name");
        Jxfs_config_key        = initProps.getProperty("Jxfs_config_key");
        Jxfs_server_port       = initProps.getProperty("Jxfs_server_port");
        Jxfs_server_name       = initProps.getProperty("Jxfs_server_name");
        Jxfs_logFilename       = initProps.getProperty("Jxfs_logFilename");
        Jxfs_traceLevel        = (new
                                  Integer(initProps.getProperty("Jxfs_traceLevel","0"))).intValue();
    }
}
```

J/XFS Endbenutzer Client (nur zu Demonstrationszwecken)

```
package com.wincornixdorf.pce.rzdiplomarbeit.jxfs.enduserclient;

// J/XFS connector
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.connector.RzProtoJxfsConnector;

// J/XFS 2.0
import com.jxfs.general.*;      // contains classes used in more than one other package
import com.jxfs.control.*;      // contains all the interfaces to J/XFS Device Controls
import com.jxfs.control.cdr.*;  // Device Control for cash dispenser, cash recycler (JxfsCashRecycler)
                                // and automatic teller machine
import com.jxfs.events.*;       // contains all the defined J/XFS events, exceptions and respective
                                // listeners

// J2SE 1.4
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;


/**
 * Enduser client for dispensing money and get information about cassette counters. <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoEndUserClient implements IJxfsOperationCompleteListener,
                                              IJxfsStatusListener {

    // connector to J/XFS
    private RzProtoJxfsConnector connector;
    // device to observe
    private String deviceName;
    // the device control
    private JxfsATM atmControl;
    // fields for gui
    private Vector cassetteFields;


    /**
     * The class constructor.
     *
     */
    public RzProtoEndUserClient() {
        System.out.println("<RzProtoEndUserClient> -> initiated ...");
        this.connector  = RzProtoJxfsConnector.getDefaultInstance();
        this.atmControl = null;
        this.deviceName = "Dummy_CashDispenser@localhost";
        this.cassetteFields = new Vector();

        this.initCashDispenser();
    }

    /**
     * send request for dispensing specified amount of money. <br>
     *
     * @param   money  amount of money
     */
    public void dispenseMoney(int money) {
```

```
        if (atmControl != null) {
            try {
                // build JxfsDispenseRequest
                Vector denominationItems = new Vector();
                JxfsDenominationItem denominationItem100 = new
                JxfsDenominationItem(0, 0);
                denominationItems.addElement(denominationItem100);

                long amount  = money*100;
                long cashBox = 0;
                JxfsDenomination denomination = new
                JxfsDenomination(denominationItems, amount, cashBox);

                int mixNumber = 0;  // test value
                int position  = 0;  // test value

                int exponent  = 0;  // test value
                String currencyString = "DEM";
                JxfsCurrencyCode currencyCode = new JxfsCurrencyCode(currencyString);
                JxfsCurrency currency = new JxfsCurrency(currencyCode, exponent);

                JxfsDispenseRequest dispenseRequest = new
                JxfsDispenseRequest(mixNumber, denomination, currency, position);

                // dispense
                System.out.println("<RzProtoEndUserClient> -> Dispensing " + money + "
                DM ...");
                int opID;
                synchronized (this) {
                    opID = atmControl.dispense(dispenseRequest);
                }
            }
            catch (JxfsException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * send an asynchron call for cassette counters. <br>
     * the result information (CashUnit) is send in an operation complete event.
     *
     *
     */
    public void queryCashUnits() {
        if (atmControl != null) {
            try {
                int opID;
                synchronized (this) {
                    opID = atmControl.queryCashUnit();
                }
            }
            catch (JxfsException e) {
                e.printStackTrace();
            }
        }
    }


    /**
     * Listener interface method to handle OperationCompleteEvents. <br>
     * get information about cash units if "JXFS_O_CDR_QUERY_CASH_UNIT" occurred. <br>
```

```
     * if cash unit is empty generate and fire a StatusEvent. <br>
     * build cassette counters fields for GUI. <br>
     *
     * @param   event an operation complete event
     */
    public synchronized void operationCompleteOccurred(JxfsOperationCompleteEvent event) {

        // if event "QUERY CASH UNIT" occurred
        if (event.getOperationID() == IJxfsCDRConst.JXFS_O_CDR_QUERY_CASH_UNIT) {
            // get cash units
            JxfsCashUnit cashUnits = (JxfsCashUnit) (event.getData());
            // get logical units in the cash unit
            Vector logicalCashUnits = cashUnits.getLogicalUnits();
            // get information of every cash unit
            System.out.println("\nKassetten:");
            for(int i=0;i<logicalCashUnits.size();i++) {
                JxfsLogicalCashUnit lcu = (JxfsLogicalCashUnit)
                logicalCashUnits.elementAt(i);

                String  unitId          = lcu.getUnitID();
                int     value           = lcu.getCashType().getValue();
                int     count           = lcu.getCount();
                int     initialCount    = lcu.getInitialCount();

                JxfsThresholdStatus ts  = lcu.getThresholdStatus();
                boolean isEmpty         = ts.isEmpty();

                System.out.println("Geldkassette mit " + (value/100) + " DM Scheinen:");
                System.out.println("Anzahl Scheine: " + count);

                // if cash unit is empty generate status event
                // JxfsStatusEvent: source, status
                if (isEmpty) {
                    JxfsStatusEvent statusEvent = new JxfsStatusEvent(atmControl,
                    JxfsThresholdStatus.JXFS_S_BIN_EMPTY);
                    try {
                        atmControl.fireStatusEvent(statusEvent);
                    }
                    catch (Exception e) {
                        e.printStackTrace();
                    }
                }

                // build text representation for gui
                String cassetteInfo =   "   " + i + "\t  " + unitId +
                                        "\t          " + (value/100) +
                                        "\t              " + count;
                this.cassetteFields.addElement(cassetteInfo);
            }
        }
    }

    /**
     * Listener interface method to handle StatusEvents. <br>
     *
     * @param   event an status event
     */
    public synchronized void statusOccurred(JxfsStatusEvent event) {
        System.out.println("Cassette empty!");
    }

    /**
```

```
     * this method returns the cassette counters fields for the GUI. <br>
     *
     * @return    vector of cassette counters for GUI representation
     */
    public Vector getCassetteFields() {
        // workareound for asynchronous call
        while (this.cassetteFields.size() != 4);
        Vector res = (Vector) this.cassetteFields.clone();
        cassetteFields.clear();
        return res;
    }

    /**
     * get device control of cash dispenser, open it <br>
     * and add operation complete listener to get information about cassettes.
     */
    private void initCashDispenser() {
        connector.startup();

        atmControl = null;
        try {
            IJxfsBaseControl c      =
            connector.getJxfsDeviceManager().getDevice(deviceName);
            JxfsBaseControl  control = (JxfsBaseControl) c;
            atmControl = (JxfsATM) control;
            System.out.println("<RzProtoEndUserClient> -> Device Control of <" +
            deviceName + "> gained ...");
        }
        catch (JxfsException exception) {
            System.out.println("<RzProtoEndUserClient> -> No device of the given type can
            found\n" + exception);
        }

        // add operation complete listener to get information about cassettes
        if (atmControl != null) {
            atmControl.addOperationCompleteListener(this);
            atmControl.addStatusListener(this);
        }

        // open ATM
        System.out.println("<RzProtoEndUserClient> -> Open <" + deviceName + "> ...");
        try {
            synchronized (this) {
                int openRes = atmControl.open();
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * MAIN for testing purposes. <br>
     *
     * @param   args command line arguments
     */
    public static void main(String[] args) {
        RzProtoEndUserClient client = new RzProtoEndUserClient();

        System.out.println("---------- End-Benutzer-Applikation ----------");
        // user input
        String inputStr="";  // user input
```

```
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        // STDIN

        while(!(inputStr.equals("quit")||inputStr.equals("exit"))) {
            System.out.print("\nGeben Sie den gewünschten Betrag ein:\n" + "> ");

            // get user input
            try {
                inputStr = stdIn.readLine();
            }
            catch(IOException iox) {
                inputStr = "";
            }

            if (!inputStr.equals("")) {
                int x;
                try {
                    x = Integer.valueOf(inputStr).intValue();
                }
                catch (Exception e) {
                    continue;
                }
                if (x%10 == 0) {
                    client.dispenseMoney(x);
                    client.queryCashUnits();
                    // wait for operation complete
                    try {
                        Thread.sleep(500);
                    }
                    catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }

        // close STDIN
        try {
            stdIn.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jxfs.eventgenerator;

// J/XFS connector
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.connector.RzProtoJxfsConnector;
// RzProtoEventForwarder MBean
import com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans.RzProtoEventForwarder;

// J/XFS 2.0
import com.jxfs.general.*;      // contains classes used in more than one other package
import com.jxfs.control.*;      // contains all the interfaces to J/XFS Device Controls
import com.jxfs.events.*;       // contains all the defined J/XFS events, exceptions and respective
listeners
import com.jxfs.control.cdr.*;  // Device Control for cash dispenser, cash recycler (JxfsCashRecycler)
and automatic teller machine
// J2SE 1.4
import java.util.*;


/**
 * Dummy class to get "CashUnit empty" events from dummy device services. <br>
 * The problem that is fixed with this class is that device services, <br>
 * that are not connected to real hardware (dummies) can not create StatusEvents! <br>
 * This class periodically checks the cassette counters (thread) of a cash dispenser <br>
 * and if a CashUnit is empty creates an StatusEvent and delivers it to the RzProtoEventForwarder.
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoJxfsEventGenerator extends Thread
                                        implements IJxfsOperationCompleteListener {

    // shows the status of this thread
    private boolean active;
    // connector to J/XFS
    private RzProtoJxfsConnector connector;
    // device to observe
    private String deviceName;
    // the device control
    private JxfsATM atmControl;
    // event listener MBean
    private RzProtoEventForwarder listener;
    // vector of IDs of actually empty cash units
    private Vector emptyLogicalCashUnits;


    /**
     * The class constructor. <br>
     * initialize properties and the J/XFS connector.
     *
     * @param   listener listener that is an event forwarder
     * @param   deviceName name of CashDispenser device
     */
    public RzProtoJxfsEventGenerator(RzProtoEventForwarder listener, String deviceName) {
        System.out.println("<RzProtoJxfsEventGenerator> -> initiated ...");
        this.connector  = RzProtoJxfsConnector.getDefaultInstance();
        this.active     = false;
        this.deviceName = deviceName;
        this.listener   = listener;
        this.emptyLogicalCashUnits = new Vector();
    }
```

```
	/**
	 * thread starter method, periodically check cassette counters. <br>
	 * if a CashUnit is empty inform the listener (RzProtoEventForwarder).
	 *
	 */
	public void run() {
		while (active) {
			this.queryCashUnits();
			try {
				this.sleep(5000);
			}
			catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}

	/**
	 * starts the connector and the polling thread.
	 *
	 */
	public void startup() {
		connector.startup();
		this.initCashDispenser();
		this.active = true;
		this.start();
	}

	/**
	 * stops the connector and the polling thread.
	 *
	 */
	public void shutdown() {
		Vector controls = new Vector();
		controls.add(atmControl);
		connector.shutdown(controls);
		this.active = false;
	}

	/**
	 * Listener interface method to handle OperationCompleteEvents. <br>
	 * get information about cash units if "IJXFS_O_CDR_QUERY_CASH_UNIT" occurred. <br>
	 *
	 * @param   event an operation complete event
	 *
	 */
	public synchronized void operationCompleteOccurred(JxfsOperationCompleteEvent event) {

		// if event "QUERY CASH UNIT" occurred
		if (event.getOperationID() == IJxfsCDRConst.JXFS_O_CDR_QUERY_CASH_UNIT) {
			// get cash units
			JxfsCashUnit cashUnits = (JxfsCashUnit) (event.getData());
			// get logical units in the cash unit
			Vector logicalCashUnits = cashUnits.getLogicalUnits();
			// get information of every cash unit
			for(int i=0;i<logicalCashUnits.size();i++) {
				JxfsLogicalCashUnit lcu = (JxfsLogicalCashUnit)
				logicalCashUnits.elementAt(i);

				JxfsThresholdStatus ts  = lcu.getThresholdStatus();
				boolean isEmpty         = ts.isEmpty();
```

```
				// if cash unit is empty generate status event
				// JxfsStatusEvent: source, status, details (LogicalCashUnit)
				if (isEmpty && (!emptyLogicalCashUnits.contains(lcu.getUnitID()))) {
					emptyLogicalCashUnits.addElement(lcu.getUnitID());
					JxfsStatusEvent statusEvent = new JxfsStatusEvent(atmControl,
					JxfsThresholdStatus.JXFS_S_BIN_EMPTY, lcu);
					try {
						listener.statusOccurred(statusEvent);
					}
					catch (Exception e) {
						e.printStackTrace();
					}
				}
			}
		}
	}

	/**
	 * get device control of cash dispenser, open it
	 * and add operation complete listener to get information about cassettes
	 */
	private void initCashDispenser() {
		connector.startup();
		// get device control of cash dispenser
		atmControl = null;
		try {
			IJxfsBaseControl c      =
			connector.getJxfsDeviceManager().getDevice(deviceName);
			JxfsBaseControl  control = (JxfsBaseControl) c;
			atmControl = (JxfsATM) control;
			System.out.println("<RzProtoJxfsEventGenerator> -> Device Control of <" +
			deviceName + "> gained ...");
		}
		catch (JxfsException exception) {
			System.out.println("<RzProtoJxfsEventGenerator> -> No device of the given type
			can be found\n" + exception);
		}

		// add operation complete listener to get information about cassettes
		if (atmControl != null) {
			atmControl.addOperationCompleteListener(this);
		}

		// open ATM
		System.out.println("<RzProtoJxfsEventGenerator> -> Open <" + deviceName + "> ...");
		try {
			synchronized (this) {
				int openRes = atmControl.open();
			}
		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}



	/**
	 * send an asynchron call for cassette counters
	 * the information is send in an operation complete event
```

```
	 */
	private void queryCashUnits() {
		if (atmControl != null) {
			try {
				int opID;
				synchronized (this) {
					opID = atmControl.queryCashUnit();
				}
			}
			catch (JxfsException e) {
				e.printStackTrace();
			}
		}
	}
}
```

J/XFS Logik

```
package com.wincornixdorf.pce.rzdiplomarbeit.jxfs.statusreader;

// J/XFS to Schema Mapper
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.mapper.RzProtoJxfsToSchemaMapper;
// J/XFS connector
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.connector.RzProtoJxfsConnector;
// device status schema for castor framework
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.devicestatusschema.*;

// J/XFS 2.0
import com.jxfs.general.*;      // contains classes used in more than one other package
import com.jxfs.control.*;      // contains all the interfaces to J/XFS Device Controls
import com.jxfs.events.*;       // contains all the defined J/XFS events, exceptions and respective
listeners
import com.jxfs.control.cdr.*;  // Device Control for cash dispenser, cash recycler (JxfsCashRecycler)
and automatic teller machine
// J2SE 1.4
import java.util.Vector;


/**
 * Class to read the status information about a CashDispenser via J/XFS. <br>
 * This class also checks if a device with a given name exists in the J/XFS environment. <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 *
 */
public class RzProtoJxfsStatusReader {

	// connector to J/XFS
	private RzProtoJxfsConnector connector;
	// the device control
	private JxfsATM atmControl;
	// name of device
	private String deviceName;


	/**
	 * The class constructor. <br>
	 *
	 */
	public RzProtoJxfsStatusReader() {
		System.out.println("<RzProtoJxfsStatusReader> -> initiated ...");
		this.connector  = RzProtoJxfsConnector.getDefaultInstance();
		this.atmControl = null;
		this.deviceName = null;
	}

	/**
	 * checks if device with the given name exists in the J/XFS environment. <br>
	 *
	 * @param   deviceName name of device
	 * @return    true if device exists, otherwise false
	 */
	public boolean isDeviceExisting(String deviceName) {
		this.deviceName = deviceName;
		try {
			this.initCashDispenser();
```

```
                return true;
            }
            catch (JxfsException exception) {
                System.out.println("<RzProtoJxfsStatusReader> -> No device of the given type
                can be found!\n" + exception);
                return false;
            }
        }

        /**
         * retrieves actual status of device with given name from J/XFS. <br>
         * then maps J/XFS information to DeviceStatus schema class and returns info.
         *
         * @param   deviceName name of device
         * @return    status of device as DeviceStatus schema class
         */
        public DeviceStatus getStatus(String deviceName) {
            this.deviceName = deviceName;
            try {
                this.initCashDispenser();
            }
            catch (JxfsException exception) {
                System.out.println("<RzProtoJxfsStatusReader> -> No device of the given type
                can be found!\n" + exception);
                return null;
            }

            DeviceStatus status = null;
            JxfsStatus jxfsStatus = null;
            try {
                jxfsStatus = atmControl.getStatus();
            }
            catch (Exception e) {
                e.printStackTrace();
            }

            // check for failure
            if (jxfsStatus == null) {
                System.out.println("<RzProtoJxfsStatusReader> -> Query for status failed!");
            }
            else {
                // map J/XFS cassette information to schema information
                System.out.println("<RzProtoJxfsStatusReader> -> Query for status successful
                ...");
                System.out.println("<RzProtoJxfsStatusReader> -> Mapping <JxfsStatus> to
                <DeviceStatus> ...");

                // build castor object
                status = RzProtoJxfsToSchemaMapper.getDeviceStatus(jxfsStatus);
            }

            System.out.println("<RzProtoJxfsStatusReader> -> Returning <DeviceStatus> ...");
            return status;
        }

        /**
         * starts the cassette counters reader. <br>
         * starts the connector.
         *
         */
        public void startup() {
            connector.startup();
        }

        /**
         * shutdown the cassette counters reader. <br>
         * stop the connector.
         *
         */
        public void shutdown() {
            Vector controls = new Vector();
            controls.add(atmControl);
            connector.shutdown(controls);
            this.connector  = null;
            this.atmControl = null;
        }

        /**
         * initialize cash dispenser.
         */
        private void initCashDispenser() throws JxfsException {
            // get device control of cash dispenser
            IJxfsBaseControl c       = connector.getJxfsDeviceManager().getDevice(deviceName);
            JxfsBaseControl  control = (JxfsBaseControl) c;
            atmControl = (JxfsATM) control;
        }
    }
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jxfs.cassettecountersreader;

// J/XFS to Schema Mapper
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.mapper.RzProtoJxfsToSchemaMapper;
// J/XFS connector
import com.wincornixdorf.pce.rzdiplomarbeit.jxfs.connector.RzProtoJxfsConnector;
// cassette counters schema for castor framework
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.cashunitschema.*;

// J/XFS 2.0
import com.jxfs.general.*;      // contains classes used in more than one other package
import com.jxfs.control.*;      // contains all the interfaces to J/XFS Device Controls
import com.jxfs.events.*;       // contains all the defined J/XFS events, exceptions and respective
listeners
import com.jxfs.control.cdr.*;  // Device Control for cash dispenser, cash recycler (JxfsCashRecycler)
and automatic teller machine
// J2SE 1.4
import java.util.*;


/**
 * Class to read the cassette counter of a CashDispenser via J/XFS. <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoCassetteCountersReader  {

    // connector to J/XFS
    private RzProtoJxfsConnector connector;
    // name of device
    private String deviceName;
    // the device control
    JxfsATM atmControl;
    // timeout waiting for the event
    private long timeout;               // <--- config
    // operation complete listener to get information inherited by an event
    RzProtoOperationCompleteListener listener;


    /**
     * The class constructor.
     *
     */
    public RzProtoCassetteCountersReader() {
        System.out.println("<RzProtoCassetteCountersReader> -> initiated ...");
        this.timeout    = 5000;
        this.connector  = RzProtoJxfsConnector.getDefaultInstance();
        this.listener   = null;
        this.deviceName = null;
        this.atmControl = null;
    }

    /**
     * open CashDispenser and send cassette counters request. <br>
     * then wait until notified of an operation complete event by the listener or timeout. <br>
     * get cassette information from listener, feed the mapper and return info. <br>
     *
     * @param   deviceName name of device
     * @return    cassette counters of device (as CashUnit schema class)
     */
```

```
    public CashUnit getCassettes(String deviceName) {
        this.deviceName = deviceName;
        try {
            this.initCashDispenser();
        }
        catch (JxfsException e) {
            System.out.println("<RzProtoCassetteCountersReader> -> No device of given
            name found!\n" + e);
            return null;
        }

        CashUnit cassetteList = null;
        try {
            int openRes = atmControl.open();
            System.out.println("<RzProtoCassetteCountersReader> -> Open <" + deviceName
            + "> ...");
            int opID    = atmControl.queryCashUnit();
            System.out.println("<RzProtoCassetteCountersReader> -> Query cash units of <"
            + deviceName + "> ...");
        }
        catch (JxfsException e) {
            System.out.println(e);
            return null;
        }

        // wait until notified or timeout
        try {
            synchronized (this) {
                this.wait(timeout);
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        // get information about cassettes
        JxfsCashUnit cassetteInfo = listener.getCashUnit();

        // check for failure
        if (cassetteInfo == null) {
            System.out.println("<RzProtoCassetteCountersReader> -> Query for cassette
            counters failed!");
        }
        else {
            // map J/XFS cassette information to SSOP cassette information
            System.out.println("<RzProtoCassetteCountersReader> -> Query for cassette
            counters successful ...");
            cassetteList = RzProtoJxfsToSchemaMapper.getCassetteList(cassetteInfo);
        }
        System.out.println("<RzProtoCassetteCountersReader> -> Returning <CashUnit> ...");
        return cassetteList;
    }

    /**
     * wakes up the cassette counters reader after event occurred. <br>
     *
     */
    public synchronized void wakeup() {
        this.notify();
    }

    /**
```

```
	 * starts the cassette counters reader. <br>
	 * starts the connector.
	 *
	 */
	public void startup() {
		connector.startup();
	}

	/**
	 * shutdown the cassette counters reader. <br>
	 * stop the connector.
	 */
	 public void shutdown() {
		Vector controls = new Vector();
		controls.add(atmControl);
		connector.shutdown(controls);
		this.connector  = null;
		this.atmControl = null;
		this.deviceName = null;
		this.listener   = null;
	}

	/**
	 * initialize cash dispenser and add operation complete listener. <br>
	 * to get information about cassettes in an asynchron event.
	 */
	private void initCashDispenser() throws JxfsException {
		// get device control of cash dispenser
		IJxfsBaseControl c       = connector.getJxfsDeviceManager().getDevice(deviceName);
		JxfsBaseControl  control = (JxfsBaseControl) c;
		atmControl = (JxfsATM) control;

		// add operation complete listener to get information about cassettes
		listener = new RzProtoOperationCompleteListener(this);
		atmControl.addOperationCompleteListener(listener);
	}
}
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jxfs.cassettecountersreader;

// J/XFS 2.0
import com.jxfs.general.*;      // contains classes used in more than one other package
import com.jxfs.control.*;      // contains all the interfaces to J/XFS Device Controls
import com.jxfs.events.*;       // contains all the defined J/XFS events, exceptions and respective
                                //listeners
import com.jxfs.control.cdr.*;  // Device Control for cash dispenser, cash recycler (JxfsCashRecycler)
                                //and automatic teller machine
// J2SE 1.4
import java.util.Vector;


/**
 * Listener to get J/XFS operation complete events. <br>
 * the event "JXFS_O_CDR_QUERY_CASH_UNIT" inherits the cassettes information. <br>
 * if event arrives inform the listener. <br>
 * then the listener can retrieve the cassette counters information.
 *
 * @author   Roger Zacharias
 * @version  1.0
 */
public class RzProtoOperationCompleteListener implements IJxfsOperationCompleteListener {

	// the reader to inform of events and deliver cassette info
	private RzProtoCassetteCountersReader reader;
	// actual info of cassette units
	private JxfsCashUnit cashUnit;


	/**
	 * The class constructor. <br>
	 *
	 * @param   reader    the PvJxfsCassCountReader to inform about event
	 */
	public RzProtoOperationCompleteListener(RzProtoCassetteCountersReader reader) {
		this.reader   = reader;
		this.cashUnit = null;
	}

	/**
	 * return actual info about cassette units if event occurred. <br>
	 *
	 * @return    CashUnit object (already mapped to schema class)
	 */
	public JxfsCashUnit getCashUnit() {
		return this.cashUnit;
	}

	/**
	 * Listener interface method to handle OperationCompleteEvents. <br>
	 * if event "JXFS_O_CDR_QUERY_CASH_UNIT" occurred get information about
	 * cassettes and store it in cassetteInfo. <br>
	 * then notify the reader and tell him the information is available. <br>
	 *
	 * @param   event     a operation complete event fired if a J/XFS operation is completed
	 *
	 */
	public synchronized void operationCompleteOccurred(JxfsOperationCompleteEvent event) {
		// if event "QUERY CASH UNIT" occurred
		if (event.getOperationID() == IJxfsCDRConst.JXFS_O_CDR_QUERY_CASH_UNIT) {
```

```
			// get cash units
			this.cashUnit = (JxfsCashUnit) (event.getData());
			// wake up the reader
			reader.wakeup();

			/************************ trick ****************************************************/
			// problem and solution: Dummy Device Services can´t generate status events ->
			// check threshold status manuelly and generate status event manuelly

			// get logical units in the cash unit
			Vector logicalCashUnits = cashUnit.getLogicalUnits();
			for(int i=0;i<logicalCashUnits.size();i++) {
				JxfsLogicalCashUnit lcu = (JxfsLogicalCashUnit)
				logicalCashUnits.elementAt(i);
				JxfsThresholdStatus ts  = lcu.getThresholdStatus();
				boolean isEmpty         = ts.isEmpty();

				// if cash unit is empty generate status event (JxfsStatusEvent: source,
				status)
				// and inform event listener
				if (isEmpty) {
					JxfsStatusEvent statusEvent = new
					JxfsStatusEvent(event.getSource(),
					JxfsThresholdStatus.JXFS_S_BIN_EMPTY);
					reader.statusOccurred(statusEvent);
				}
			}
			/************************ trick ****************************************************/
		}
	}
}
```

J/XFS nach XML Schema Mapper

```
package com.wincornixdorf.pce.rzdiplomarbeit.jxfs.mapper;

// cassette counters schema for castor framework
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.cashunitschema.*;
// device status schema for castor framework
import com.wincornixdorf.pce.rzdiplomarbeit.schemata.devicestatusschema.*;

// J/XFS 2.0
import com.jxfs.general.*;      // contains classes used in more than one other package
import com.jxfs.control.cdr.*;  // Device Control for cash dispenser, cash recycler (JxfsCashRecycler)
and automatic teller machine
// J2SE 1.4
import java.util.Vector;


/**
 * Class to map J/XFS information (cassette counters, device state) to
 * schema classes. <br>
 *
 * @author   Roger Zacharias
 * @version  1.0
 *
 */
public class RzProtoJxfsToSchemaMapper {

	/**
	 * returns a CashUnit object out of given JxfsCashUnit
	 *
	 * @param   jxfsCassetteInfo a JxfsCashUnit
	 * @return   the mapped CashUnit schema class
	 */
	public static CashUnit getCassetteList(JxfsCashUnit jxfsCassetteInfo) {
		System.out.println("<RzProtoJxfsToSchemaMapper> -> Mapping <JxfsCashUnit> to
		<CashUnit> ...");
		CashUnit cu  = new CashUnit();
		Vector logicalCashUnits = jxfsCassetteInfo.getLogicalUnits();

		// get information of every cash unit
		for (int i=0; i<logicalCashUnits.size(); i++) {
			JxfsLogicalCashUnit lcu  = (JxfsLogicalCashUnit) logicalCashUnits.elementAt(i);

			int    number         = lcu.getNumber();          // cassette number
			String  unitId        = lcu.getUnitID();          // cassette id
			int    value          = lcu.getCashType().getValue(); // denomination
			int    initialCount   = lcu.getInitialCount();    // initial number
			int    dispenseCount  = lcu.getDispenseCount();   // number of dispensed
			int    rejectCount    = lcu.getRejectCount();     // number of rejected
			int    count          = lcu.getCount();           // number of actual
			int    cuType         = lcu.getCuType();          // type
			int    cuKind         = lcu.getCuKind();          // type
			int    status         = lcu.getStatus();          // state
			boolean appLock       = lcu.isAppLock();          // state
			boolean devLock       = lcu.isDevLock();          // state
			JxfsCashType cashType  = lcu.getCashType();       // type
			// currency
			String  currencyCode   = cashType.getCurrencyCode().getCurrencyCode();
			JxfsThresholdStatus ts  = lcu.getThresholdStatus();   // threshold status
			boolean isEmpty        = ts.isEmpty();             // state
			boolean isFull         = ts.isFull();              // state
```

```
                boolean isHigh      = ts.isHigh();          // state
                boolean isLow       = ts.isLow();           // state

                // get fitting type
                String type   = getType(cuType,cuKind);
                // get fitting state
                String state  = getState(status,appLock,devLock,isEmpty,isFull,isHigh,isLow);

                // build castor cash unit
                LogicalCashUnit tmp = new LogicalCashUnit();
                tmp.setNumber(number);
                tmp.setId(unitId);
                tmp.setValue(value);
                tmp.setInitialCount(initialCount);
                tmp.setDispenseCount(dispenseCount);
                tmp.setRejectCount(rejectCount);
                tmp.setCount(count);
                tmp.setCurrencyCode(currencyCode);
                tmp.setType(type);
                tmp.setStatus(state);
                cu.addLogicalCashUnit(tmp);
        }
        System.out.println("<RzProtoJxfsToSchemaMapper> -> Mapping successfull ...");
        return cu;
    }

    /**
     * returns a DeviceStatus object out of given JxfsCashUnit
     *
     * @param   jxfsCassetteInfo a JxfsCashUnit
     * @return  the mapped CashUnit schema class
     */
    public static DeviceStatus getDeviceStatus(JxfsStatus jxfsStatus) {
        System.out.println("<RzProtoJxfsToSchemaMapper> -> Mapping <JxfsDeviceStatus> to
        <DeviceStatus> ...");
        DeviceStatus status = new DeviceStatus();
        status.setBusy(jxfsStatus.isBusy());
        status.setClaimed(jxfsStatus.isClaimed());
        status.setClaimPending(jxfsStatus.isClaimPending());
        status.setHardwareError(jxfsStatus.isHardwareError());
        status.setOpen(jxfsStatus.isOpen());
        status.setPowerSave(jxfsStatus.isPowerSave());
        status.setUserActionError(jxfsStatus.isUserActionError());
        status.setWorking(jxfsStatus.isWorking());
        System.out.println("<RzProtoJxfsToSchemaMapper> -> Mapping successfull ...");
        return status;
    }

    /**
     * build type out of the given parameters.
     */
    private static String getType(int cuType, int cuKind) {
        String res = null;
        if ((cuType==IJxfsCDRConst.JXFS_C_CDR_LCU_BILL_CASSETTE) &&
        (cuKind==IJxfsCDRConst.JXFS_C_CDR_LCU_DISPENSE)) {
                res = "CassetteTypeNormal";
        }
        else if (cuType==IJxfsCDRConst.JXFS_C_CDR_LCU_REJECT_CASSETTE) {
                res = "CassetteTypeReject";
        }
        else if (cuType==IJxfsCDRConst.JXFS_C_CDR_LCU_RETRACT_CASSETTE) {
                res = "CassetteTypeRetract";
        }
        else if ((cuType==IJxfsCDRConst.JXFS_C_CDR_LCU_COIN_CYLINDER) &&
                (cuKind==IJxfsCDRConst.JXFS_C_CDR_LCU_DEPOSIT)) {
                res = "CassetteTypeCoinIn";
        }
        else if (cuType==IJxfsCDRConst.JXFS_C_CDR_LCU_COIN_DISPENSER) {
                res = "CassetteTypeCoinOut";
        }
        else if ((cuType==IJxfsCDRConst.JXFS_C_CDR_LCU_BILL_CASSETTE) &&
        (cuKind==IJxfsCDRConst.JXFS_C_CDR_LCU_DEPOSIT)) {
                res = "CassetteTypeNotesIn";
        }
        else if ((cuType==IJxfsCDRConst.JXFS_C_CDR_LCU_BILL_CASSETTE) &&
        (cuKind==IJxfsCDRConst.JXFS_C_CDR_LCU_RECYCLE)) {
                res = "CassetteTypeRoller";
        }
        else if ((cuType==IJxfsCDRConst.JXFS_C_CDR_LCU_COIN_CYLINDER) &&
                (cuKind==IJxfsCDRConst.JXFS_C_CDR_LCU_RECYCLE)) {
                res = "CassetteTypeCoinInOut";
        }
        else {
                res = "Unknown";
        }
        return res;
    }

    /**
     * build state out of the given parameters.
     */
    private static String getState(int status,boolean appLock, boolean devLock,
            boolean isEmpty, boolean isFull,
            boolean isHigh, boolean isLow) {
            String res = null;
            if (status==IJxfsCDRConst.JXFS_C_CDR_LCU_OK) {
                res = "ok";
            }
            else if (status==IJxfsCDRConst.JXFS_C_CDR_LCU_MISSING) {
                res = "missing";
            }
            else if (devLock) {
                res = "locked";
            }
            else if (isEmpty) {
                res = "empty";
            }
            else if (isLow) {
                res = "low";
            }
            else if (isFull) {
                res = "full";
            }
            else if (isHigh) {
                res = "high";
            }
            else if (appLock) {
                res = "logical locked";
            }
            else {
                res = "unknown";
            }
            return res;
    }
}
```

C# SOAP-Client

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace RzProtoOperatorClientDotNet
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.RichTextBox richTextBox1;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        private System.Windows.Forms.Button button3;

        // *** Web Service Proxy ***
        localhost4.RzProtoOperatorWebServiceSimpleService service;

        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //
            // TODO: Add any constructor code after InitializeComponent call
            //

            // *** initialize Web Service Proxy ***
            service = new localhost4.RzProtoOperatorWebServiceSimpleService();
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
```

```
        }
        base.Dispose( disposing );
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.richTextBox1 = new System.Windows.Forms.RichTextBox();
        this.label4 = new System.Windows.Forms.Label();
        this.label5 = new System.Windows.Forms.Label();
        this.label1 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.label3 = new System.Windows.Forms.Label();
        this.textBox2 = new System.Windows.Forms.TextBox();
        this.textBox1 = new System.Windows.Forms.TextBox();
        this.button1 = new System.Windows.Forms.Button();
        this.button2 = new System.Windows.Forms.Button();
        this.button3 = new System.Windows.Forms.Button();
        this.SuspendLayout();
        //
        // richTextBox1
        //
        this.richTextBox1.Location = new System.Drawing.Point(8, 184);
        this.richTextBox1.Name = "richTextBox1";
        this.richTextBox1.Size = new System.Drawing.Size(352, 136);
        this.richTextBox1.TabIndex = 8;
        this.richTextBox1.Text = "";
        //
        // label4
        //
        this.label4.Location = new System.Drawing.Point(0, 360);
        this.label4.Name = "label4";
        this.label4.Size = new System.Drawing.Size(368, 23);
        this.label4.TabIndex = 5;
        this.label4.Click += new System.EventHandler(this.label4_Click);
        //
        // label5
        //
        this.label5.Font = new System.Drawing.Font("Microsoft Sans Serif", 9.75F,
        System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
        ((System.Byte)(0)));
        this.label5.Location = new System.Drawing.Point(8, 160);
        this.label5.Name = "label5";
        this.label5.Size = new System.Drawing.Size(136, 23);
        this.label5.TabIndex = 6;
        this.label5.Text = "Kommandoauswahl";
        this.label5.Click += new System.EventHandler(this.label5_Click);
        //
        // label1
        //
        this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 9.75F,
        System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
        ((System.Byte)(0)));
        this.label1.Location = new System.Drawing.Point(8, 16);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(104, 24);
        this.label1.TabIndex = 0;
        this.label1.Text = "Geräteauswahl";
        this.label1.Click += new System.EventHandler(this.label1_Click);
        //
        // label2
        //
        this.label2.Location = new System.Drawing.Point(8, 56);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(80, 23);
        this.label2.TabIndex = 1;
        this.label2.Text = "Gerätename";
        this.label2.Click += new System.EventHandler(this.label2_Click);
        //
        // label3
        //
        this.label3.Location = new System.Drawing.Point(8, 80);
        this.label3.Name = "label3";
        this.label3.Size = new System.Drawing.Size(80, 23);
        this.label3.TabIndex = 1;
        this.label3.Text = "Hostname";
        this.label3.Click += new System.EventHandler(this.label3_Click);
        //
        // textBox2
        //
        this.textBox2.Location = new System.Drawing.Point(112, 80);
        this.textBox2.Name = "textBox2";
        this.textBox2.Size = new System.Drawing.Size(176, 20);
        this.textBox2.TabIndex = 3;
        this.textBox2.Text = "localhost";
        this.textBox2.TextChanged += new
        System.EventHandler(this.textBox2_TextChanged);
        //
        // textBox1
        //
        this.textBox1.Location = new System.Drawing.Point(112, 56);
        this.textBox1.Name = "textBox1";
        this.textBox1.Size = new System.Drawing.Size(176, 20);
        this.textBox1.TabIndex = 2;
        this.textBox1.Text = "CashDispenser";
        this.textBox1.TextChanged += new
        System.EventHandler(this.textBox1_TextChanged);
        //
        // button1
        //
        this.button1.Location = new System.Drawing.Point(184, 112);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(104, 23);
        this.button1.TabIndex = 4;
        this.button1.Text = "Gerät wählen!";
        this.button1.Click += new System.EventHandler(this.button1_Click);
        //
        // button2
        //
        this.button2.Location = new System.Drawing.Point(8, 328);
        this.button2.Name = "button2";
        this.button2.Size = new System.Drawing.Size(136, 23);
        this.button2.TabIndex = 7;
        this.button2.Text = "Statusinformation";
        this.button2.Click += new System.EventHandler(this.button2_Click);
        //
        // button3
        //
        this.button3.Location = new System.Drawing.Point(224, 328);
        this.button3.Name = "button3";
        this.button3.Size = new System.Drawing.Size(136, 23);
        this.button3.TabIndex = 7;
        this.button3.Text = "Kassetteninformation";
        this.button3.Click += new System.EventHandler(this.button3_Click);
        //
        // Form1
        //
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.ClientSize = new System.Drawing.Size(368, 381);
        this.Controls.AddRange(new System.Windows.Forms.Control[] {
                this.button3,
                this.richTextBox1,
                this.button2,
                this.label5,
                this.label4,
                this.button1,
                this.textBox2,
                this.textBox1,
                this.label3,
                this.label2,
                this.label1});
        this.Name = "Form1";
        this.Text = "Form1";
        this.Load += new System.EventHandler(this.Form1_Load);
        this.ResumeLayout(false);

    }
    #endregion

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.Run(new Form1());
    }

    private void label3_Click(object sender, System.EventArgs e)
    {

    }

    private void textBox2_TextChanged(object sender, System.EventArgs e)
    {

    }

    private void button1_Click(object sender, System.EventArgs e)
    {
        // *** use web service ***
        String device = this.textBox1.Text + "@" + this.textBox2.Text;
        bool result = service.isDeviceExisting(device);
        if (result)
        {
            this.label4.Text = "Zugriff auf Gerät <" + device + "> erfolgreich!";
        }
        else
        {
            this.label4.Text = "Gerät <" + device + "> existiert nicht!";
        }
    }

    private void label2_Click(object sender, System.EventArgs e)
    {

    }

    private void textBox1_TextChanged(object sender, System.EventArgs e)
    {

    }

    private void label1_Click(object sender, System.EventArgs e)
    {

    }

    private void Form1_Load(object sender, System.EventArgs e)
    {

    }

    private void label4_Click(object sender, System.EventArgs e)
    {

    }

    private void label5_Click(object sender, System.EventArgs e)
    {

    }

    private void button2_Click(object sender, System.EventArgs e)
    {
        // *** use web service ***
        String device = this.textBox1.Text + "@" + this.textBox2.Text;
        String result = service.getDeviceStatus(device);
        this.label4.Text = "Statusinformationen erfolgreich erhalten!";
        this.richTextBox1.Text = result;
    }

    private void button3_Click(object sender, System.EventArgs e)
    {
        // *** use web service ***
        String device = this.textBox1.Text + "@" + this.textBox2.Text;
        String result = service.getCassCount(device);
        this.label4.Text = "Kassetteninformationen erfolgreich erhalten!";
        this.richTextBox1.Text = result;
    }
}
```